Semantic Properties of Asynchronous Orc

David Kitchin, William R. Cook and Jayadev Misra *
The University of Texas at Austin
{dkitchin, wcook, misra}@cs.utexas.edu

January 15, 2007

Abstract

Orc is a new language for task orchestration, a form of concurrent programming with applications in workflow, business process management, and web service orchestration. Orc provides constructs to orchestrate the concurrent invocation of services – while managing time-outs, priorities, and failure of services or communication. In this paper, we show a trace-based semantic model for Orc, which induces a congruence on Orc programs and facilitates reasoning about them. Despite the simplicity of the language and its semantic model, Orc is able to express a variety of useful orchestration tasks.

1 Introduction

We describe the semantic properties of *Orc*, a new language for *task orchestration*. Task orchestration is a form of concurrent programming in which multiple services are invoked to achieve a goal – while managing time-outs, priorities, and failures of services and communication. Unlike traditional concurrency models, orchestration introduces an *asymmetric* relationship between a program and the services that constitute its environment. An orchestration invokes and receives responses from the external services, which do not initiate communication.

Many practical problems can be understood as orchestrations – for example, business workflows are naturally expressed as orchestrations [**?**]. We illustrate the use of Orc in implementing some traditional concurrent computation patterns; larger examples have also been developed [**?**, **?**]. Orc has also been used to study service-level agreements for composite web services [**?**].

The goal of this work is to develop a language based on a simple trace semantics that can still express, and support reasoning about, useful task orchestrations. The key metric for a trace semantics is whether trace equivalence corresponds to observational equivalence of programs. Depending on the language, traces must be extended to include failures, refusals, ready states, etc.,

*Work of the second author is partially supported by National Science Foundation grant CCF-0448128.

in order to adequately model observational equivalence. The trace semantics of Orc is a simple set of traces; the traces include communication events and substitution events, which model synchronization. We prove that the equality of trace sets defines a congruence on programs, in that programs with equivalent trace sets are interchangeable. We show a number of laws about Orc programs, similar to those in Kleene algebra [**?**]; the laws are based on strong bisimulation. We then introduce a more general congruence based on trace set equivalence which can establish laws not provable by strong bisimulation.

2 Overview of Orc

An Orc program consists of a set of definitions and a *goal* expression which is to be evaluated. Orc assumes that basic services, like sequential computation and data manipulation, are implemented by primitive *sites*. Orc provides constructs to orchestrate the concurrent invocation of sites.

The syntax of Orc is given in Figure 1. A site call $M(\bar{p})$ invokes a site named M with a list of actual parameters $\bar{p}$. If there are no parameters, the site call is written as just 'M'. An actual parameter p may be a variable x or a value v. Calls to defined expressions $E(\bar{p})$ are similar, given a definition with name E and formal parameters $\bar{p}$. There are only three combinators: $>x>$ for sequential composition, $\mid$ for parallel composition, and **where** for asymmetric parallel composition. The combinators are listed in decreasing order of precedence, so $f >x> g \mid h$ means $(f >x> g) \mid h$, and f **where** $x :\in g \mid h$ means f **where** $x :\in (g \mid h)$. In the remainder of this section, we give an informal overview of the Orc programming model with examples. The formal semantics is given in Section 3.

2.1 Site Calls

The simplest Orc expression is a *site call* $M(\bar{p})$, where M is a site name and $\bar{p}$ is a list of actual parameters. A site is a separately defined procedure, like a web service. The site may be implemented on the client's machine or a remote machine. A site call elicits at most one response; it is possible that a site never responds to a call. For example, a site call $CNN(d)$, where CNN is a news service and d is a date, might download the newspage for the specified date. Site calls are *strict*, i.e., a site is called only if all its parameters have values.

$$
\begin{array}{rcll}
f, g, h & \in & \textit{Expression} & ::= \quad M(\bar{p}) \mid E(\bar{p}) \mid f >x> g \mid f \mid g \mid f \textbf{ where } x :\in g \\
p & \in & \textit{Actual} & ::= \quad x \mid v \\
 & & \textit{Definition} & ::= \quad E(\bar{x}) \triangleq f
\end{array}
$$

Figure 1: Syntax of Orc

$let(x, y, \cdots)$	Returns argument values as a tuple.
$if(b)$	Returns a signal if b is true, and does not respond if b is false.
$Rtimer(t)$	Returns a signal after exactly t time units.

Table 1: Fundamental Sites

Table 1 defines a few sites that are fundamental to effective programming in Orc (a *signal* is a value which has no additional information). Additionally, **0** represents a site which never responds.

2.2 Combinators

There are three combinators in Orc to compose expressions. Symmetric composition of f and g, written as $f \mid g$, evaluates f and g independently. The sites called by f and g are the ones called by $f \mid g$ and any value published by either f or g is published by $f \mid g$. There is no direct communication or interaction between these two computations. For example, evaluation of $CNN(d) \mid BBC(d)$ initiates two independent computations; up to two values will be published depending on the number of sites that respond.

In $f >x> g$, expression f is evaluated, each value published by it initiates a fresh instance of g as a separate computation, and the value published by f is called x in g's computation. Evaluation of f continues while (possibly several) instances of g are run. This is the only mechanism in Orc similar to spawning threads. If f publishes no value, g is never instantiated. The values published by the executions of different instances of g are the values published by $f >x> g$. As an example, the following expression calls sites *CNN* and *BBC* in parallel to get the news for date d. Responses from either of these calls are bound to x and then site *email* is called to send the information to address a. Thus, *email* may be called 0, 1 or 2 times.

$$(CNN(d) \mid BBC(d)) >x> email(a, x)$$

Expression $f \gg g$ is a short-hand for $f >x> g$ when x is not used in g.

To evaluate $(f \textbf{ where } x :\in g)$, start by evaluating both f and g in parallel. Evaluation of parts of f which do not depend on x can proceed, but site calls in which x is a parameter are suspended until x has a value. In $((M \mid N(x)) \textbf{ where } x :\in R)$, for example, M can be called even before x has a value. If g publishes a value, then x is assigned this value, g's evaluation is terminated and the suspended parts of f can proceed. This is the only mechanism in Orc to block or terminate parts of a computation. Unlike in the previous example, the following expression calls *email* at most once.

$$email(a, x) \textbf{ where } x :\in (CNN(d) \mid BBC(d))$$

2.3 Definitions

Declaration $E(\bar{x}) \triangleq f$ defines expression E whose formal parameter list is $\bar{x}$ and body is expression f. A call $E(\bar{p})$ is evaluated by replacing the formal parameters $\bar{x}$ by the actual parameters $\bar{p}$ in the body of the definition f. Sites are called by value, while definitions are called by name.

2.4 Examples

Time-out

Expression $let(z)$ **where** $z :\in (f \mid Rtimer(t) \gg let(3))$ either publishes the first publication of f, or times out after t units and publishes 3. A typical programming paradigm is to call site M and publish a pair (x, b) as the value, where b is true if M publishes x before the time-out, and false if there is a time-out. In the latter case, x is irrelevant. Below, z is the pair (x, b).

$$let(z) \ \textbf{where}\ z :\in (\ M >x> let(x, true) \ \mid \ Rtimer(t) >x> let(x, false)\ )$$

Fork-join Parallelism

In concurrent programming, one often needs to spawn two independent threads at a point in the computation, and resume the computation after both threads complete. Such an execution style is called *fork-join* parallelism. There is no special construct for fork-join in Orc, but it is easy to code such computations. The following code fragment calls sites M and N in parallel and publishes their values as a tuple after they both complete their executions.

$$(let(u, v) \ \textbf{where}\ u :\in M) \ \textbf{where}\ v :\in N$$

Synchronization

There is no special machinery for synchronization in Orc; a **where** expression provides the necessary ingredients for programming synchronizations. Consider $M \gg f$ and $N \gg g$; we wish to execute them independently, but synchronize f and g by starting them only after *both* M and N have completed.

$$((let(u, v) \ \textbf{where}\ u :\in M) \ \textbf{where}\ v :\in N) \gg (f \mid g)$$

Priority

Call the sites M and N, but give priority to M by publishing its response if it arrives within the first time unit, even if N's response precedes it.

$$let(x) \ \textbf{where}\ x :\in (M \mid ((Rtimer(1) \gg let(u)) \ \textbf{where}\ u :\in N))$$

Arbitration

A fundamental problem in concurrent computing is *arbitration*: to choose between two computations and let only one proceed. Arbitration is the essence of mutual exclusion. Consider a process which behaves as process P if event *Alpha* happens and as Q if *Beta* happens. In Orc, events *Alpha* and *Beta* are represented as sites, and P and Q are expressions. Below, *flag* records which of *Alpha* and *Beta* responds first.

$$\begin{array}{l} \mathit{if}(\mathit{flag}) \gg P \mid \mathit{if}(\neg \mathit{flag}) \gg Q \\ \quad \textbf{where} \;\; \mathit{flag} :\in (\mathit{Alpha} \gg \mathit{let}(\mathit{true}) \mid \mathit{Beta} \gg \mathit{let}(\mathit{false})) \end{array}$$

The Orc model permits more complex arbitration protocols, such as: execute one of P, Q and R, depending how many sites out of *Alpha*, *Beta* and *Gamma* respond within 10 time units.

Recursive definitions of expressions

The recursive expression *Metronome*, defined below, publishes a signal every time unit, starting immediately. It is used in the subsequent expression to call site *Poll* each time unit and publish its values.

$$\begin{array}{l} \mathit{Metronome} \;\; \underline{\Delta} \;\; \mathit{Signal} \mid \mathit{Rtimer}(1) \gg \mathit{Metronome} \\ \mathit{Metronome} \gg \mathit{Poll} \end{array}$$

Non-strict Evaluation; Parallel-or

Parallel-or is a classic problem in non-strict evaluation: computation of $x \vee y$ over booleans x and y publishes *true* if either variable value is *true*; therefore, the evaluation may terminate even when one of the variable values is unknown. Here, we state the problem in Orc terms, and give a simple solution.

Suppose sites M and N publish booleans. Compute the *parallel-or* of the two booleans, i.e., (in a non-strict fashion) publish *true* as soon as either site returns *true* and *false* only if both sites return *false*. In the following solution, site $\mathit{or}(x, y)$ returns $x \vee y$. Site $\mathit{ift}(b)$ returns *true* if b is true; it does not respond otherwise: $\mathit{ift}(b) = \mathit{if}(b) \gg \mathit{let}(\mathit{true})$.

$$\begin{array}{l} (\;\; (\mathit{let}(z) \;\textbf{where}\; z :\in \mathit{ift}(x) \mid \mathit{ift}(y) \mid \mathit{or}(x, y)\,) \\ \qquad\qquad \textbf{where}\; x :\in M \;\;) \\ \qquad\qquad\quad \textbf{where}\; y :\in N \end{array}$$

3 Asynchronous Semantics

We develop a formal semantics of Orc in this section. The semantics is operational, asynchronous, and based on labeled transition systems. A synchronous semantics has also been defined [**?**], while a complete temporal semantics is future work.

As is common in small-step operational semantics, the syntax of Orc must be extended to represent intermediate states. We introduce $?k$ to denote an instance of a site call that has not yet returned a value, where k is a unique handle that identifies the call instance. We also restrict the language to sites and definitions with a single argument; multiple arguments are easily handled by adding tuples to the language.

The transition relation $f \xrightarrow{a} f'$, defined in Figure 2, states that expression f may transition with event a to expression f'. There are four kinds of events, which we call *base events*:

$$a, b \in BaseEvent ::= !v \mid \tau \mid M_k(v) \mid k?v$$

A *publication* event, $!v$, publishes a value v from an expression. As is traditional, τ denotes an *internal* event. The remaining two events, the *site call* event $M_k(v)$ and the *response* event $k?v$, are discussed below.

3.1 Site Calls

A site call involves three steps: invocation of the site, response from the site, and publication of the result. These steps can be arbitrarily interleaved with other site calls, or delayed indefinitely.

Rule SiteCall specifies that a site call $M(v)$, where v is a value, transitions to $?k$ with event $M_k(v)$. The handle k connects a site call to a site return – a fresh handle is created for each call to identify that call instance. The resulting expression, $?k$, represents a process that is blocked waiting for the response from the call. A site call occurs only when its parameters are values; in $M(x)$, where x is a variable, the call is blocked until x is defined.

In SiteRet a pending site call $?k$ receives a result v from the environment and transitions to the expression $let(v)$. There is no assumption that all site calls eventually respond. If there is no response, then the call blocks indefinitely.

The Let rule generates a publication event $!v$ from its argument value v.

3.2 Composition Rules

The composition rules are straightforward, except in some cases where subexpressions publish values. When f publishes a value ($f \xrightarrow{!v} f'$), rule Seq1V creates a new instance of the right side, $[v/x].g$, the expression in which all free occurrences of x in g are replaced by v. The publication $!v$ is hidden, and the entire expression performs a τ action. Note that f and all instances of g are executed in parallel. Because the semantics is asynchronous, there is no guarantee that the values published by the first instance will precede the values of later instances. Instead, the values produced by all instances of g are interleaved arbitrarily.

Asymmetric parallel composition is similar to parallel composition, except when g publishes a value v. In this case, rule Asym1V terminates g and x is bound to v in f. One subtlety of these rules is that f may contain both active and blocked subprocesses – any site call that uses x is blocked until g publishes.

$$\frac{k \text{ fresh}}{M(v) \xrightarrow{M_k(v)} ?k} \text{(SiteCall)} \qquad \frac{f \xrightarrow{a} f' \quad a \neq !v}{f >x> g \xrightarrow{a} f' >x> g} \text{(Seq1N)}$$

$$?k \xrightarrow{k?v} let(v) \text{ (SiteRet)} \qquad \frac{f \xrightarrow{!v} f'}{f >x> g \xrightarrow{\tau} (f' >x> g) \mid [v/x].g} \text{(Seq1V)}$$

$$let(v) \xrightarrow{!v} \mathbf{0} \text{ (Let)}$$

$$\frac{f \xrightarrow{a} f'}{f \mid g \xrightarrow{a} f' \mid g} \text{(Sym1)} \qquad \frac{f \xrightarrow{a} f'}{f \textbf{ where } x :\in g \xrightarrow{a} f' \textbf{ where } x :\in g} \text{(Asym1N)}$$

$$\frac{g \xrightarrow{a} g'}{f \mid g \xrightarrow{a} f \mid g'} \text{(Sym2)} \qquad \frac{g \xrightarrow{!v} g'}{f \textbf{ where } x :\in g \xrightarrow{\tau} [v/x].f} \text{(Asym1V)}$$

$$\frac{(E(x) \triangleq f) \in D}{E(p) \xrightarrow{\tau} [p/x].f} \text{(Def)} \qquad \frac{g \xrightarrow{a} g' \quad a \neq !v}{f \textbf{ where } x :\in g \xrightarrow{a} f \textbf{ where } x :\in g'} \text{(Asym2)}$$

Figure 2: Asynchronous Operational Semantics of Orc

Expressions are evaluated using call-by-name in the Def rule. We assume a single global set of definitions D.

4 Executions and Traces

Define the relation $\Rightarrow$ as the transitive closure of the transition relation $\rightarrow$, together with the empty transition ϵ. If $f \stackrel{s}{\Rightarrow} f'$, we say that s is an *execution* of f.

$$f \stackrel{\epsilon}{\Rightarrow} f \qquad \frac{f \xrightarrow{a} f'', \; f'' \stackrel{s}{\Rightarrow} f'}{f \stackrel{as}{\Rightarrow} f'}$$

We write as to denote the concatenation of event a onto execution s. Similarly, st is the concatenation of two executions s and t. We have included $f \stackrel{\epsilon}{\Rightarrow} f$ to guarantee that the set of executions of an expression is prefix-closed.

A *trace* is obtained by removing all internal events, τ, from an execution. Note that every execution (and trace) is finite in length. However, we can represent an infinite sequence of transitions by the set of all of its finite prefixes.

Example

An execution of $((M(x) \mid let(x)) >y> R(y))$ **where** $x :\in (N \mid S)$ is shown below. In the following, N returns value 5 and $R(5)$ returns 7.

$$S_k \quad N_l \quad l?5 \quad M_m(5) \quad \tau \quad R_n(5) \quad n?7 \quad !7$$

The response from S, if any, is ignored after N responds. The event τ is due to $let(5) \stackrel{!5}{\rightarrow} \mathbf{0}$. This event causes the event $R_n(5)$. Site call $M_m(5)$ has not yet responded in this execution. The final expression is $(?m >y> R(y))$.

4.1 Laws proved using Strong Bisimulation

A *closed* expression is one which has no free variables; an *open* expression has free variables. In this section, we list certain identities over closed expressions, some of them similar to the laws of Kleene algebra [**?**]. We write $f \sim g$ to denote that f and g are strongly bisimilar [**?**]. In a later section, we develop a notion of congruence over both closed and open expressions. That notion can be used to show, for example, that $f >x> let(x)$ is congruent to f.

Below, "f is x-free" means that x is not a free variable of f.

1. $f \mid \mathbf{0} \sim f$
2. $f \mid g \sim g \mid f$
3. $f \mid (g \mid h) \sim (f \mid g) \mid h$
4. $f >x> (g >y> h) \sim (f >x> g) >y> h$, if h is x-free.
5. $\mathbf{0} >x> f \sim \mathbf{0}$
6. $(f \mid g) >x> h \sim f >x> h \mid g >x> h$
7. $(f \mid g)$ **where** $x :\in h \sim (f$ **where** $x :\in h) \mid g$, if g is x-free.
8. $(f >y> g)$ **where** $x :\in h \sim (f$ **where** $x :\in h) >y> g$, if g is x-free.
9. $(f$ **where** $x :\in g)$ **where** $y :\in h \sim (f$ **where** $y :\in h)$ **where** $x :\in g$, if g is y-free and h is x-free.
10. $\mathbf{0}$ **where** $x :\in ?k \sim ?k \gg \mathbf{0}$
11. $\mathbf{0}$ **where** $x :\in M \sim M \gg \mathbf{0}$, for any site M.

4.2 Substitution Events

Strong bisimulation is applicable only if each side of an identity is a closed expression. If we relax this restriction, we can use bisimulation to prove, for example, that $\mathbf{0} = let(x)$, because neither has a non-trivial transition. Yet, these two expressions display different behaviors in the same context. For example, $let(1) >x> \mathbf{0}$ never publishes whereas $let(1) >x> let(x)$ always publishes.

To obtain a more general theory we introduce a new kind of event, a *substitution event* of the form $[v/x]$, and the transition rule:

$$f \stackrel{[v/x]}{\rightarrow} [v/x].f \qquad \text{(SUBST)}$$

A substitution event differs from the base events described in Section 3 in a crucial way: the rules in Figure 2 are defined only over base events. Therefore, given that $f \stackrel{[v/x]}{\rightarrow} [v/x].f$, (SYM1) can *not* be applied to deduce

$$f \mid g \stackrel{[v/x]}{\rightarrow} [v/x].f \mid g,$$

Allowing substitution events expands the set of executions (and traces) of expressions. For example, the traces of $let(x)$ are of the form $[v/x]\ !v$, for all possible v, and their prefixes. Introducing substitution events allows us to distinguish between $\mathbf{0}$ and $let(x)$, for instance, because a possible trace of $let(x)$ is $[1/x]\ !1$.

We observe that proofs by strong bisimulation of the laws of Section 4.1 remain valid after allowing for substitution events. This is because both sides of an identity $f \sim g$ are closed. Hence, given $f \stackrel{a}{\rightarrow} f'$, either a is not a substitution event, or it is of the form $[v/x]$ where x is not free in f. In the latter case, $f' = f$, and this transition corresponds in g to $g \stackrel{a}{\rightarrow} g$.

Furthermore, we can now show that all the laws of Section 4.1 hold for arbitrary expressions f, g, and h, open or closed. So, we can prove, for instance, that $let(x) \mid let(y) \sim let(y) \mid let(x)$. Each side of an identity has the same set of free variables (except $\mathbf{0} >x> f \sim \mathbf{0}$, which is easily handled). Therefore, a proof by strong bisimulation applies to each identity.

Variable Naming

Free and bound variables of an expression have different names. Hence, in $(f >x> g)$, f does not have a free variable x, and in $(f \textbf{ where } x :\in g)$, x is not free in g.

5 Characterizations of Traces

Notation

We write $\langle f \rangle$ for the set of traces of f.

In this section, we show that the traces of an expression can be determined from the traces of its constituent subexpressions. In particular, we overload the Orc combinators to apply over trace sets and prove that

$\langle f \mid g \rangle = \langle f \rangle \mid \langle g \rangle$
$\langle f >x> g \rangle = \langle f \rangle >x> \langle g \rangle$
$\langle f \textbf{ where } x :\in g \rangle = \langle f \rangle \textbf{ where } x :\in \langle g \rangle$

5.1 Trace characterization of base expressions

Notation

We use the following notation for quantified expressions: $(\cup r : r \in R : e)$ denotes $\cup_{r \in R}(e)$, where variable r can be free in e. Range R of r may be omitted if it is clear from context, e.g., $(\cup i :: S_i)$.

We show the trace sets of base expressions, i.e, those without constituent subexpressions. We only list the compact versions of traces in which there are no substitutions to irrelevant variables (which have no effect on the rest of the trace). Also, we only list the maximal traces below, whose prefixes constitute the entire trace set. Below, *Values* denotes the set of all possible responses from sites.

$$\langle let(v) \rangle = \{\, !v\}$$
$$\langle M(v) \rangle = (\cup w : w \in \mathit{Values} : \{M_k(v)\ k?w\ \ !w\})$$
$$\langle M(x) \rangle = (\cup v : v \in \mathit{Values} : (\cup t : t \in \langle M(v) \rangle : \{[v/x]t\}))$$

The trace set for $let(v)$ is easy to see. For $M(v)$, any maximal trace is of the form $M_k(v)\ k?w\ \ !w$, where w is a response from M. Note that k is a bound parameter of the trace (with $M_k(v)$ as its binder) and it can be renamed consistently. The trace set of $M(x)$ starts with a substitution $[v/x]$, for any v, followed by any trace of $M(v)$.

5.2 Trace characterization for $(f \mid g)$

Separation and Merge

Let s, t and p be sequences of events. We call p a *merge* of s and t if (1) s and t are both subsequences of p and every event of p belongs to at least one of s and t, (2) every common event of p (i.e., an event that belongs to both s and t) is a substitution, and (3) for any variable which has a substitution in both s and t, its first substitution in both s and t is a common event of p. We call the pair $(s,\ t)$ a *separation* of p.

Example

Let

$$\begin{array}{lcl} s & = & a \quad b \quad [3/x] \quad [4/x] \quad [5/x] \\ t & = & c \quad [2/y] \quad [3/x] \quad [5/x] \quad [4/x] \\ u & = & [3/x] \quad [2/y] \end{array}$$

Below, we use subscripts on events to identify the sequences to which they belong, when there is ambiguity.

$$\begin{array}{llllllll} a & c & b & [2/y] & [3/x]_{s,t} & [4/x]_{s} & [5/x]_{s,t} & [4/x]_{t} \quad \in (s \mid t) \\ a & b & [3/x]_{s,u} & [4/x] & [5/x] & [2/y] & & \quad \in (s \mid u) \end{array}$$

There is no merge for t and u because the orders of first substitutions to x and y are different. Also, if two sequences have $[v/x]$ and $[w/x]$ as their first substitutions for x, and $v \neq w$, then they have no merge, from condition (3). Note that in the merge of s and t, $[5/x]$ appears once, whereas $[4/x]$ appears twice. Condition (3) imposes a constraint only on the first substitution to a variable; subsequent substitutions may or may not be common events in a merge.

Definition

For traces s and t, define $s \mid t$ to be the set of their merges.

$$s \mid t = \{p \mid p \text{ is a merge of } s \text{ and } t\}$$

We lift the definition to trace sets $S \mid T$:

$$S \mid T = \{p \mid p \in s \mid t,\ s \in S,\ t \in T\}$$

Note that $s \mid \epsilon = \{s\}$, $S \mid \{\epsilon\} = S$, and $\mid$ over traces is commutative.

Theorem 1 $\langle f \mid g \rangle = \langle f \rangle \mid \langle g \rangle$

Proof Sketch: The complete proof is in Appendix C.1. The proof is in two parts showing that one side is a subset of the other. For $\langle f \mid g \rangle \subseteq \langle f \rangle \mid \langle g \rangle$, we show a separation (s, t) of any trace p of $\langle f \mid g \rangle$ such that $s \in \langle f \rangle$ and $t \in \langle g \rangle$. The proof is by induction on the length of p. In the other direction, to show $\langle f \rangle \mid \langle g \rangle \subseteq \langle f \mid g \rangle$, let p be a trace with separation (s, t) where $s \in \langle f \rangle$ and $t \in \langle g \rangle$. We prove that $p \in \langle f \mid g \rangle$ by induction on the length of p.

5.3 Trace characterization for $(f >x> g)$

Define operator $\backslash$ as follows: $T\backslash[v/x] = \{t \mid [v/x]t \in T\}$. That is, $T\backslash[v/x]$ discards all traces in T that do not begin with $[v/x]$, and removes the leading $[v/x]$ event from the remaining traces.

We extend this notation to sequences of substitutions:

$T\backslash\epsilon = T$
$T\backslash(cD) = (T\backslash c)\backslash D$,
where c is a substitution and D a sequence of substitutions.

Definition

For trace s and trace set T define a set of traces $(s >x> T)$ by

$$\begin{cases} \{s\} & \text{if } s \text{ has no publication,} \\ r(t >x> T' \mid (T'\backslash[v/x])) & \text{if } s = r\ !v\ t \text{ and } r \text{ has no publication,} \\ \quad \text{where } D \text{ is the sequence of substitutions in } r, & \\ \quad \text{and } T' = T\backslash D & \end{cases}$$

We lift the definition to $S >x> T$, where S and T are sets of traces.

$$S >x> T = \{p \mid p \in s >x> T,\ s \in S\}$$

Theorem 2 $\langle f >x> g\rangle = \langle f\rangle >x> \langle g\rangle$

Proof Sketch: The complete proof is in Appendix C.2. The proof is in two parts showing that one side is a subset of the other. To prove that $\langle f\rangle >x> \langle g\rangle \subseteq \langle f >x> g\rangle$, take any p which is in $\langle f\rangle >x> \langle g\rangle$, i.e., $p \in (s >x> \langle g\rangle)$, where $s \in \langle f\rangle$. Then show, by induction on the number of publications in s, that $p \in \langle f >x> g\rangle$. In the other direction, to show $\langle f >x> g\rangle \subseteq \langle f\rangle >x> \langle g\rangle$, we take a trace p of $\langle f >x> g\rangle$, and construct a sequence s which corresponds to the subsequence of events from f in p. We prove, by induction on the number of publications in s, that $p \in (s >x> \langle g\rangle)$.
Note: Any substitution event $[v/x]$ in s is unrelated to x in $(s >x> T)$.

5.4 Trace characterization for $(f$ **where** $x :\in g)$

Definition

For traces s and t define a set of traces $(s$ **where** $x :\in t)$ by

$$\begin{cases} (s \mid t) & \text{if } t \text{ has no publication,} \\ (s' \mid t')s'' & \text{if } s = s'\ [v/x]\ s'',\ t = t'\ !v\ t'', \\ & t' \text{ has no publication and } s' \text{ has no substitution for } x \\ \{\} & \text{otherwise .} \end{cases}$$

We lift the definition to $(S$ **where** $x :\in T)$, where S and T are sets of traces.

$$(S \textbf{ where } x :\in T) = \{p \mid p \in (s \textbf{ where } x :\in t),\ s \in S,\ t \in T\}$$

Theorem 3 $\langle f$ **where** $x :\in g\rangle = \langle f\rangle$ **where** $x :\in \langle g\rangle$

Proof Sketch: The complete proof is in Appendix C.3. The proof is in two parts showing that one side is a subset of the other. If g never publishes, rule (ASYM1V) is never used; therefore, the operational behavior of $(f$ **where** $x :\in g)$ is analogous that of $f \mid g$ because (ASYM1N) and (ASYM2) are the counterparts of (SYM1) and (SYM2), respectively. Then, any trace of $(f$ **where** $x :\in g)$ is from $(s \mid t)$, where s and t are traces of f and g. If g has a trace $t'\ !v\ t''$ and f a trace $s'\ [v/x]\ s''$, then $(s' \mid t')s''$ is a trace of $(f$ **where** $x :\in g)$, using the above argument and the meaning of substitution.
Note: Any substitution event $[v/x]$ in t is unrelated to x in $(s$ **where** $x :\in t)$.

6 Monotonicity and Continuity of Combinators

The results of the last section show that the set of traces of any expression can be obtained from the traces of its constituent subexpressions. This motivates the following definition of *congruence* among expressions: two expressions are congruent, $\cong$, if their trace sets are equal. Similarly, define a partial order, $\sqsubseteq$, over expressions.

$f \cong g$ if $\langle f \rangle = \langle g \rangle$, and $f \sqsubseteq g$ if $\langle f \rangle \subseteq \langle g \rangle$

Each combinator preserves substitution of congruent subexpressions. That is, given $f \cong g$, we claim

1. $f \mid h \cong g \mid h$, and $h \mid f \cong h \mid g$
2. $f >x> h \cong g >x> h$, and $h >x> f \cong h >x> g$
3. f **where** $x :\in h \cong g$ **where** $x :\in h$, and
   h **where** $x :\in f \cong h$ **where** $x :\in g$

We prove the results by showing that each combinator is monotonic in both its arguments. That is, given $f \sqsubseteq g$, we prove the claims (1, 2, 3) with $\sqsubseteq$ replacing $\cong$. Then switching the roles of f and g, we get the congruences. We also prove continuity of the combinators. Underlying most of the proofs is the following lemma.

Lemma 4 Let each of $S_0, S_1, \cdots$ and T be a set of traces. Then,

1. $(\cup i :: S_i * T) = (\cup i :: S_i) * T$, where $*$ is any Orc combinator.
2. $(\cup i :: T * S_i) = T * (\cup i :: S_i)$, where $*$ is any combinator other than $>x>$.

Proof: From the definition of $*$ over trace sets, for arbitrary R and T,

$R * T = (\cup r : r \in R : r * T)$, where $*$ is any Orc combinator
$R * T = (\cup t : t \in T : R * t)$, where $*$ is any combinator other than $>x>$.

These follow from the lifting in the definition of the combinators over sets.

6.1 Monotonicity of Orc combinators

Each Orc combinator is monotonic in its left argument, e.g. $f \sqsubseteq g$ implies $f \mid h \sqsubseteq g \mid h$. This follows from Lemma 4, part(1); see Appendix D.1 for details.

Monotonicity in the right argument for combinators other than $>x>$, (e.g. $f \sqsubseteq g$ implies $(h$ **where** $x :\in f) \sqsubseteq (h$ **where** $x :\in g)$), follows from Lemma 4, part(2). We give a proof that $f \sqsubseteq g$ implies $(h >x> f) \sqsubseteq (h >x> g)$ in Appendix D.1.

6.2 Continuity of Orc combinators

Characterization of Least Upper Bound

Let f denote a sequence of expressions $f_0, f_1, \cdots$, where $f_i \sqsubseteq f_{i+1}$, for all i. Expression F is an upper bound of f if for all i, $f_i \sqsubseteq F$, and F is the least upper bound of f if for any upper bound G of f, $F \sqsubseteq G$. Henceforth, we write $(\sqcup f)$ for the least upper bound of f. The proof of the following theorem is standard, and is given in Appendix D.2.

Theorem 5 $\langle \sqcup f \rangle = (\cup i :: \langle f_i \rangle)$.

Continuity over left argument

Let f be a sequence of expressions, and $h_i = f_i * g$, for all i, where $*$ is any Orc combinator. Then

$$\sqcup h \ \cong \ (\sqcup f) * g, \text{ i.e., } \langle \sqcup h \rangle = \langle (\sqcup f) * g \rangle.$$

The proof follows directly from Lemma 4, part(1).

Continuity over right argument

Given a sequence g, and $h_i = f * g_i$, for all i, where $*$ is any Orc combinator, we show $\sqcup h \ \cong \ f * (\sqcup g)$. The proof follows directly from Lemma 4, part(2), where $*$ is any combinator other than $>x>$. For $>x>$, we prove the result in Appendix D.2.

6.3 Least fixed point for recursive definitions

We have shown that $\sqsubseteq$ is a complete partial order over expressions. Next, we show that $\mathbf{0}$ is the bottom element. Any substitution $[v/x]$ is applicable to any expression because $f \stackrel{[v/x]}{\rightarrow} [v/x].f$. Hence, any sequence of substitutions D is a trace of any f (by applying induction on the length of D). Since $\mathbf{0}$ has no other transition,

$$\langle \mathbf{0} \rangle = \{D | \ D \text{ is a finite sequence of substitutions}\}$$

Therefore, for any f, $\langle \mathbf{0} \rangle \subseteq \langle f \rangle$, i.e., $\mathbf{0} \sqsubseteq f$.

Monotonicity and continuity of Orc combinators allow us to treat a recursively defined expression as the least upper bound of a chain of approximations. As an example, consider *Metronome* (described in Section 2.4) which we repeat below in an abbreviated form.

$$M \triangleq S \mid R \gg M$$

Then M is the least upper bound of the chain $M_0 \sqsubseteq M_1 \sqsubseteq \cdots$, where

$M_0 = \mathbf{0}$
$M_{i+1} = S \mid R \gg M_i$, for all i, where $i \geq 0$

6.4 A proof using congruence

Theorem 6 $f \ >x> \ let(x) \ \cong \ f$

Proof Sketch: The complete proof is in Appendix E. The proof is by structural induction on f. For the base cases (i.e. when f is any of $let(p)$, $?k$, $M(p)$, or $M(x)$), the result is proved by enumerating the traces (all maximal traces of f are of the form $r \ !v$, where r has no publications). For the inductive case, we consider three possible forms of f which are: $g \mid h$, $g \ >y> \ h$ and g **where** $y :\in h$. We apply certain laws from Section 4.1. From law (6),

$(g \mid h) >x> let(x)$ is $g >x> let(x) \mid h >x> let(x)$, and inductively, this is $g \mid h$. From law (4), $(g >y> h) >x> let(x)$ is $g >y> (h >x> let(x))$, which is $g >y> h$, using induction on $h >x> let(x)$. From law (8), $(g \textbf{ where } y :\in h) >x> let(x)$ is $(g >x> let(x)) \textbf{ where } y :\in h$, which is $g \textbf{ where } y :\in h$, using induction on $g >x> let(x)$.

7 Related Work

There are two primary bodies of work on developing models for task orchestration. On the one hand, commercial workflow and orchestration languages have been the subject of formal study. On the other hand, traditional process algebra theory is being adapted to fit orchestration problems.

Petri nets have been proposed as a semantic model for workflow [**?**]. To compare commercial systems, Van der Aalst has proposed a set of workflow patterns [**?**]. These workflow patterns have been also been implemented in Orc [**?**] and the π-calculus [**?**]. Others have identified difficult patterns, like time-out [**?**], which have solutions in Orc.

A new Petri net language, YAWL, has been defined to express the patterns more directly [**?**]. YAWL's mechanism for multiple instantiation is analogous to Orc's sequential composition, but provides built-in synchronization. The node grouping and cancellation constructs are similar to Orc's **where** operator. Rather than build specific workflow patterns into the language, Orc provides just a few fundamental primitives with a mechanism to define new operators for user-defined composition patterns.

Process calculi, including CSP [**?**], CCS [**?**] and π-calculus [**?**], provide fundamental models of concurrency, with a focus on symmetric communication between threads. Orc has a structured approach to concurrency, and has an asymmetric relationship with its environment. Orc also supports a general sequential composition of expressions, $f \gg g$, and an explicit construct for process termination, which is synchronized with communication. Some variants of the π-calculus include a termination construct [**?**].

Simon Peyton-Jones suggested a relationship between Orc and the List monad as used in functional programming languages, including Haskell . The sequential composition operator, $>x>$, is analogous to the list bind >>=. The **where** operator resembles taking the first item from a lazy list. The standard list monad always produces values in a specific order, whereas the publication order in Orc is non-deterministic.

8 Conclusion

Task orchestration is a form of concurrent programming in which an agent invokes and coordinates the execution of passive, but potentially unreliable, services. Orchestration is well-suited to solving a range of concurrency problems, most notably workflow. Our practical experience in using Orc for orchestration

has been very encouraging; we are able to code most concurrent programming paradigms succinctly. This paper shows that Orc has a simple trace semantics, and Orc combinators have many desirable properties such as monotonicity and continuity. The simplicity of the semantics may be a factor in simplicity of programming. We have addressed only the asynchronous aspects of Orc in this paper. We are now developing the full semantics which will combine asynchrony with time-based computations.

A Proofs of Laws of Section 4.1

The proofs in this section employ strong bisimulation. A *strong simulation* is a binary relation $\leq$ over closed expressions with the following properties: given $f \leq g$ and $f \stackrel{a}{\rightarrow} f'$, there is a transition $g \stackrel{a}{\rightarrow} g'$ such that $f' \leq g'$. A *strong bisimulation*, $\sim$, is a strong simulation which is also a symmetric relation. To prove $f \sim g$, we prove $f \leq g$ and $g \leq f$, for some strong simulation $\leq$.

To prove $f \leq g$, for expressions f and g, we have to show that for each transition of f there is a corresponding transition of g. Each transition of f is based on some transition of a component subexpression. Therefore, we look at all component subexpressions of f (which is same as that of g, for the laws given in this section), enumerate their transitions, and show that each of those transitions have the same effect in f and g.

Note that each transition of f and g has a corresponding transition in $f \mid g$ and $(f \textbf{ where } x :\in g)$. And only a transition of f has a corresponding transition in $f >x> g$.

1. $f \mid \mathbf{0} \sim f$.

   Proof:

   The only subexpression is f. Subexpression $\mathbf{0}$ has no transition.

   $$\begin{array}{ll} & f \stackrel{a}{\rightarrow} f' \\ \Rightarrow & \{\text{Sym1}\} \\ & f \mid \mathbf{0} \stackrel{a}{\rightarrow} f' \mid \mathbf{0} \end{array}$$

   And,

   $$f \stackrel{a}{\rightarrow} f'$$

   Assumed: $f' \mid \mathbf{0} \sim f'$.

2. $f \mid g \sim g \mid f$.

   First, we consider the transitions of f.

   $$\begin{array}{ll} & f \stackrel{a}{\rightarrow} f' \\ \Rightarrow & \{\text{Sym1}\} \\ & f \mid g \stackrel{a}{\rightarrow} f' \mid g \end{array}$$

   $$\begin{array}{ll} & f \stackrel{a}{\rightarrow} f' \\ \Rightarrow & \{\text{Sym2}\} \\ & g \mid f \stackrel{a}{\rightarrow} g \mid f' \end{array}$$

   Assumed: $f' \mid g \sim g \mid f'$

   The derivation with g's transition is symmetric.

3. $f \mid (g \mid h) \sim (f \mid g) \mid h$. We consider the transitions of f, g and h in turn.

(a) (Transition of f: $f \xrightarrow{a} f'$)

$$\begin{array}{ll} & f \xrightarrow{a} f' \\ \Rightarrow & \{\text{Sym1}\} \\ & f \mid (g \mid h) \xrightarrow{a} f' \mid (g \mid h) \end{array}$$

And,

$$\begin{array}{ll} & f \xrightarrow{a} f' \\ \Rightarrow & \{\text{Sym1}\} \\ & f \mid g \xrightarrow{a} f' \mid g \\ \Rightarrow & \{\text{Sym1}\} \\ & (f \mid g) \mid h \xrightarrow{a} (f' \mid g) \mid h \end{array}$$

Assumed: $f' \mid (g \mid h) \sim (f' \mid g) \mid h$

(b) (Transition of g: $g \xrightarrow{a} g'$)

$$\begin{array}{ll} & g \xrightarrow{a} g' \\ \Rightarrow & \{\text{Sym1}\} \\ & g \mid h \xrightarrow{a} g' \mid h \\ \Rightarrow & \{\text{Sym2}\} \\ & f \mid (g \mid h) \xrightarrow{a} f \mid (g' \mid h) \end{array}$$

And,

$$\begin{array}{ll} & g \xrightarrow{a} g' \\ \Rightarrow & \{\text{Sym2}\} \\ & f \mid g \xrightarrow{a} f \mid g' \\ \Rightarrow & \{\text{Sym1}\} \\ & (f \mid g) \mid h \xrightarrow{a} (f \mid g') \mid h \end{array}$$

Assumed: $f \mid (g' \mid h) \sim (f \mid g') \mid h$

(c) (Transition of h: $h \xrightarrow{a} h'$)

$$\begin{array}{ll} & h \xrightarrow{a} h' \\ \Rightarrow & \{\text{Sym2}\} \\ & g \mid h \xrightarrow{a} g \mid h' \\ \Rightarrow & \{\text{Sym2}\} \\ & f \mid (g \mid h) \xrightarrow{a} f \mid (g \mid h') \end{array}$$

And,

$$\begin{array}{ll} & h \xrightarrow{a} h' \\ \Rightarrow & \{\text{Sym2}\} \\ & (f \mid g) \mid h \xrightarrow{a} (f \mid g) \mid h' \end{array}$$

Assumed: $f \mid (g \mid h') \sim (f \mid g) \mid h'$

4. $f >x> (g >y> h) \sim (f >x> g) >y> h$, provided h is x-free.

   Only the transitions of f have corresponding transitions in $f >x> (g >y> h)$. And, only the transitions of f have corresponding transitions in $f >x> g$, and hence in $(f >x> g) >y> h$. Therefore, we consider only the transitions of f, publications and non-publications.

(a) $(f \stackrel{!v}{\rightarrow} f')$

$$\begin{array}{ll} & f \stackrel{!v}{\rightarrow} f' \\ \Rightarrow & \{\text{Seq1V}\} \\ & f >x> (g >y> h) \stackrel{\tau}{\rightarrow} f' >x> (g >y> h) \mid [v/x].(g >y> h) \end{array}$$

And,

$$\begin{array}{ll} & f \stackrel{!v}{\rightarrow} f' \\ \Rightarrow & \{\text{Seq1V}\} \\ & f >x> g \stackrel{\tau}{\rightarrow} f' >x> g \mid ([v/x].g) \\ \Rightarrow & \{\text{Seq1N}\} \\ & (f >x> g) >y> h \stackrel{\tau}{\rightarrow} (f' >x> g \mid [v/x].g) >y> h \end{array}$$

We show $f' >x> (g >y> h) \mid [v/x].(g >y> h) \sim (f' >x> g \mid [v/x].g) >y> h$

$$\begin{array}{ll} & f' >x> (g >y> h) \mid [v/x].(g >y> h) \\ = & \{\text{substitution distributes}\} \\ & f' >x> (g >y> h) \mid ([v/x].g) >y> ([v/x].h) \\ = & \{h \text{ is } x\text{-free}\} \\ & f' >x> (g >y> h) \mid ([v/x].g) >y> h \end{array}$$

And,

$$\begin{array}{ll} & (f' >x> g \mid [v/x].g) >y> h \\ \sim & \{\text{distributivity law}\} \\ & (f' >x> g) >y> h \mid [v/x].g >y> h \\ \sim & \{\text{Associativity}\} \\ & f' >x> (g >y> h) \mid ([v/x].g) >y> h \end{array}$$

(b) $(f \stackrel{a}{\rightarrow} f', a \neq !v)$

$$\begin{array}{ll} & f \stackrel{a}{\rightarrow} f' \\ \Rightarrow & \{\text{Seq1N}\} \\ & f >x> (g >y> h) \stackrel{a}{\rightarrow} f' >x> (g >y> h) \end{array}$$

And,

$$\begin{array}{ll} & f \stackrel{a}{\rightarrow} f' \\ \Rightarrow & \{\text{Seq1N}\} \\ & f >x> g \stackrel{a}{\rightarrow} f' >x> g \\ \Rightarrow & \{\text{Seq1N}\} \\ & (f >x> g) >y> h \stackrel{a}{\rightarrow} (f' >x> g) >y> h \end{array}$$

Assumed: $f' >x> (g >y> h) \sim (f' >x> g) >y> h$, given h is x-free.

Corollary: $f \gg (g >y> h) \sim (f \gg g) >y> h$

5. $\mathbf{0} \gg f \sim \mathbf{0}$, $\mathbf{0} >x> f \sim \mathbf{0}$.

   Only transitions of $\mathbf{0} \gg f$ and $\mathbf{0}$ correspond to those of $\mathbf{0}$, and $\mathbf{0}$ has no transition.

6. $(f \mid g) >x> h \sim f >x> h \mid g >x> h$.

   We consider only the transitions of f and g because transitions of h do not have corresponding transitions for either expression. By symmetry and the commutativity of $\mid$ we need consider only the transitions of f.

   (a) $(f \xrightarrow{a} f',\ a \neq !v)$

   $$\begin{array}{ll} & f \xrightarrow{a} f' \\ \Rightarrow & \{\text{Sym1}\} \\ & f \mid g \xrightarrow{a} f' \mid g \\ \Rightarrow & \{\text{Seq1N}\} \\ & (f \mid g) >x> h \xrightarrow{a} (f' \mid g) >x> h \end{array}$$

   And,

   $$\begin{array}{ll} & f \xrightarrow{a} f' \\ \Rightarrow & \{\text{Seq1N}\} \\ & f >x> h \xrightarrow{a} f' >x> h \\ \Rightarrow & \{\text{Sym1}\} \\ & f >x> h \mid g >x> h \xrightarrow{a} f' >x> h \mid g >x> h \end{array}$$

   Assumed: $(f' \mid g) >x> h \sim f' >x> h \mid g >x> h$.

   (b) $(f \xrightarrow{!v} f')$

   $$\begin{array}{ll} & f \xrightarrow{!v} f' \\ \Rightarrow & \{\text{Seq1V}\} \\ & f >x> h \xrightarrow{\tau} f' >x> h \mid [v/x].h \\ \Rightarrow & \{\text{Sym1}\} \\ & f >x> h \mid g >x> h \xrightarrow{\tau} (f' >x> h \mid [v/x].h) \mid g >x> h \end{array}$$

   And,

   $$\begin{array}{ll} & f \xrightarrow{!v} f' \\ \Rightarrow & \{\text{Sym1}\} \\ & f \mid g \xrightarrow{!v} f' \mid g \\ \Rightarrow & \{\text{Seq1V}\} \\ & (f \mid g) >x> h \xrightarrow{\tau} (f' \mid g) >x> h \mid [v/x].h \end{array}$$

   To see $(f' >x> h \mid [v/x].h) \mid g >x> h \sim (f' \mid g) >x> h \mid [v/x].h$

   $$\begin{array}{ll} & (f' \mid g) >x> h \mid [v/x].h \\ \sim & \{\text{distributivity}\} \\ & (f' >x> h \mid g >x> h) \mid [v/x].h \\ \sim & \{\text{associativity of } \mid\} \\ & (f' >x> h \mid [v/x].h) \mid g >x> h \end{array}$$

7. $(f \mid g)$ **where** $x :\in h \sim (f$ **where** $x :\in h) \mid g$, provided g is x-free.

   There are four different kinds of transitions for each of the expressions: transitions of f, g, publication of h and non-publication of h.

(a) $f \xrightarrow{a} f'$:

$$\begin{array}{ll} & f \xrightarrow{a} f' \\ \Rightarrow & \{\text{Sym1}\} \\ & f \mid g \xrightarrow{a} f' \mid g \\ \Rightarrow & \{\text{Asym2}\} \\ & (f \mid g)\ \textbf{where}\ x :\in h \xrightarrow{a} (f' \mid g)\ \textbf{where}\ x :\in h \end{array}$$

And,

$$\begin{array}{ll} & f \xrightarrow{a} f' \\ \Rightarrow & \{\text{Asym2}\} \\ & f\ \textbf{where}\ x :\in h \xrightarrow{a} f'\ \textbf{where}\ x :\in h \\ \Rightarrow & \{\text{Sym1}\} \\ & (f\ \textbf{where}\ x :\in h) \mid g \xrightarrow{a} (f'\ \textbf{where}\ x :\in h) \mid g \end{array}$$

Assumed: $(f' \mid g)\ \textbf{where}\ x :\in h \sim (f'\ \textbf{where}\ x :\in h) \mid g$

(b) $g \xrightarrow{a} g'$:

$$\begin{array}{ll} & g \xrightarrow{a} g' \\ \Rightarrow & \{\text{definition of Sym2}\} \\ & f \mid g \xrightarrow{a} f \mid g' \\ \Rightarrow & \{\text{definition of Asym2}\} \\ & (f \mid g)\ \textbf{where}\ x :\in h \xrightarrow{a} (f \mid g')\ \textbf{where}\ x :\in h \end{array}$$

And,

$$\begin{array}{ll} & g \xrightarrow{a} g' \\ \Rightarrow & \{\text{definition of Sym2}\} \\ & (f\ \textbf{where}\ x :\in h) \mid g \xrightarrow{a} (f\ \textbf{where}\ x :\in h) \mid g' \end{array}$$

Since g is x-free, so is g'. Assumed: $(f \mid g')\ \textbf{where}\ x :\in h \sim (f\ \textbf{where}\ x :\in h) \mid g'$.

(c) $h \xrightarrow{a} h'$, $a \neq !v$:

$$\begin{array}{ll} & h \xrightarrow{a} h' \\ \Rightarrow & \{\text{Asym1N}\} \\ & (f \mid g)\ \textbf{where}\ x :\in h \xrightarrow{a} (f \mid g)\ \textbf{where}\ x :\in h' \end{array}$$

And,

$$\begin{array}{ll} & h \xrightarrow{a} h' \\ \Rightarrow & \{\text{Asym1N}\} \\ & f\ \textbf{where}\ x :\in h \xrightarrow{a} f\ \textbf{where}\ x :\in h' \\ \Rightarrow & \{\text{Sym1}\} \\ & (f\ \textbf{where}\ x :\in h) \mid g \xrightarrow{a} (f\ \textbf{where}\ x :\in h') \mid g \end{array}$$

Given g is x-free, and assumed $(f \mid g)\ \textbf{where} x :\in h' \sim (f\ \textbf{where} x :\in h') \mid g$.

(d) $h \xrightarrow{!v} h'$:

$$
\begin{array}{ll}
 & h \stackrel{!v}{\rightarrow} h' \\
\Rightarrow & \{\text{Asym1V}\} \\
 & (f \mid g)\ \mathbf{where}\ x :\in h \stackrel{\tau}{\rightarrow} [v/x].(f \mid g)
\end{array}
$$

And,

$$
\begin{array}{ll}
 & h \stackrel{!v}{\rightarrow} h' \\
\Rightarrow & \{\text{Asym1V}\} \\
 & f\ \mathbf{where}\ x :\in h \stackrel{\tau}{\rightarrow} [v/x].f \\
\Rightarrow & \{\text{Sym1}\} \\
 & (f\ \mathbf{where}\ x :\in h) \mid g \stackrel{\tau}{\rightarrow} [v/x].f \mid g
\end{array}
$$

To see that $[v/x].(f \mid g) \sim [v/x].f \mid g$, we show they are equal.

$$
\begin{array}{ll}
 & [v/x].(f \mid g) \\
= & \{\text{substitution distributes}\} \\
 & ([v/x].f) \mid ([v/x].g) \\
= & \{g \text{ is } x\text{-free}\} \\
 & ([v/x].f) \mid g
\end{array}
$$

8. $(f >y> g)\ \mathbf{where}\ x :\in h \sim (f\ \mathbf{where}\ x :\in h) >y> g$, provided g is x-free.

   The transitions of left side expression correspond to those of $(f >y> g)$ and h, i.e., of f and h. Similarly for the right side expression. We consider publication and non-publication transitions of f and h separately.

   (a) $(f \stackrel{a}{\rightarrow} f',\ a \neq !v)$

$$
\begin{array}{ll}
 & f \stackrel{a}{\rightarrow} f' \\
\Rightarrow & \{\text{Seq1N}\} \\
 & f >y> g \stackrel{a}{\rightarrow} f' >y> g \\
\Rightarrow & \{\text{Asym2}\} \\
 & (f >y> g)\ \mathbf{where}\ x :\in h \stackrel{a}{\rightarrow} (f' >y> g)\ \mathbf{where}\ x :\in h
\end{array}
$$

   And,

$$
\begin{array}{ll}
 & f \stackrel{a}{\rightarrow} f' \\
\Rightarrow & \{\text{Asym2}\} \\
 & f\ \mathbf{where}\ x :\in h \stackrel{a}{\rightarrow} f'\ \mathbf{where}\ x :\in h \\
\Rightarrow & \{\text{Seq1N}\} \\
 & (f\ \mathbf{where}\ x :\in h) >y> g \stackrel{a}{\rightarrow} (f'\ \mathbf{where}\ x :\in h) >y> g
\end{array}
$$

   Assumed: $(f' >y> g)\ \mathbf{where}\ x :\in h \sim (f'\ \mathbf{where}\ x :\in h) >y> g$.

   (b) $(f \stackrel{!v}{\rightarrow} f')$

$$
\begin{array}{ll}
 & f \stackrel{!v}{\rightarrow} f' \\
\Rightarrow & \{\text{Seq1V}\} \\
 & f >y> g \stackrel{\tau}{\rightarrow} f' >y> g \mid [v/y].g \\
\Rightarrow & \{\text{Asym2}\} \\
 & (f >y> g)\ \mathbf{where}\ x :\in h \stackrel{\tau}{\rightarrow} (f' >y> g \mid [v/y].g)\ \mathbf{where}\ x :\in h
\end{array}
$$

   And,

$$
\begin{array}{ll}
 & f \xrightarrow{!v} f' \\
\Rightarrow & \{\text{Asym2}\} \\
 & f \textbf{ where } x :\in h \xrightarrow{!v} f' \textbf{ where } x :\in h \\
\Rightarrow & \{\text{Seq1V}\} \\
 & (f \textbf{ where } x :\in h) >y> g \xrightarrow{\tau} (f' \textbf{ where } x :\in h) >y> g \mid [v/y].g
\end{array}
$$

To see that $(f' >y> g \mid [v/y].g) \textbf{ where } x :\in h \sim (f' \textbf{ where } x :\in h) >y> g \mid [v/y].g$

$$
\begin{array}{ll}
 & (f' >y> g \mid [v/y].g) \textbf{ where } x :\in h \\
\sim & \{g \text{ is } x\text{-free. So, is } [v/y].g\} \\
 & (f' >y> g \textbf{ where } x :\in h) \mid [v/y].g \\
\sim & \{\text{this law}\} \\
 & (f' \textbf{ where } x :\in h) >y> g \mid [v/y].g
\end{array}
$$

(c) $(h \xrightarrow{a} h', a \neq !v)$

$$
\begin{array}{ll}
 & h \xrightarrow{a} h' \\
\Rightarrow & \{\text{Asym1N}\} \\
 & (f >y> g) \textbf{ where } x :\in h \xrightarrow{a} (f >y> g) \textbf{ where } x :\in h'
\end{array}
$$

And,

$$
\begin{array}{ll}
 & h \xrightarrow{a} h' \\
\Rightarrow & \{\text{Asym1N}\} \\
 & f \textbf{ where } x :\in h \xrightarrow{a} f \textbf{ where } x :\in h' \\
\Rightarrow & \{\text{Seq1N}\} \\
 & (f \textbf{ where } x :\in h) >y> g \xrightarrow{a} (f \textbf{ where } x :\in h') >y> g
\end{array}
$$

Assumed: $(f >y> g) \textbf{ where } x :\in h' \sim (f \textbf{ where } x :\in h') >y> g$

(d) $(h \xrightarrow{!v} h')$

$$
\begin{array}{ll}
 & h \xrightarrow{!v} h' \\
\Rightarrow & \{\text{Asym1V}\} \\
 & (f >y> g) \textbf{ where } x :\in h \xrightarrow{\tau} [v/x].(f >y> g)
\end{array}
$$

And,

$$
\begin{array}{ll}
 & h \xrightarrow{!v} h' \\
\Rightarrow & \{\text{Asym1V}\} \\
 & f \textbf{ where } x :\in h \xrightarrow{\tau} [v/x].f \\
\Rightarrow & \{\text{Seq1N}\} \\
 & (f \textbf{ where } x :\in h) >y> g \xrightarrow{\tau} ([v/x].f) >y> g
\end{array}
$$

To see that $[v/x].(f >y> g) \sim ([v/x].f) >y> g$:

$$
\begin{array}{ll}
 & [v/x].(f >y> g) \\
= & \{\text{substitution distributes}\} \\
 & ([v/x].f) >y> ([v/x].g) \\
= & \{g \text{ is } x\text{-free. So, } [v/x].g = g\} \\
 & ([v/x].f) >y> g
\end{array}
$$

9. $(f \textbf{ where } x :\in g) \textbf{ where } y :\in h \;\sim\; (f \textbf{ where } y :\in h) \textbf{ where } x :\in g$, provided g is y-free and h is x-free.

   We have to consider the transitions corresponding to those of f , g and h. The roles of g and h are symmetric; so, we consider only the transitions of g.

   (a) $(f \xrightarrow{a} f')$

   $$\begin{array}{ll} & f \xrightarrow{a} f' \\ \Rightarrow & \{\text{Asym2}\} \\ & f \textbf{ where } x :\in g \xrightarrow{a} f' \textbf{ where } x :\in g \\ \Rightarrow & \{\text{Asym2}\} \\ & (f \textbf{ where } x :\in g) \textbf{ where } y :\in h \xrightarrow{a} (f' \textbf{ where } x :\in g) \textbf{ where } y :\in h \end{array}$$

   And,

   $$\begin{array}{ll} & f \xrightarrow{a} f' \\ \Rightarrow & \{\text{Asym2}\} \\ & f \textbf{ where } y :\in h \xrightarrow{a} f' \textbf{ where } y :\in h \\ \Rightarrow & \{\text{Asym2}\} \\ & (f \textbf{ where } y :\in h) \textbf{ where } x :\in g \xrightarrow{a} (f' \textbf{ where } y :\in h) \textbf{ where } x :\in g \end{array}$$

   Assumed: $(f' \textbf{ where } x :\in g) \textbf{ where } y :\in h \;\sim\; (f' \textbf{ where } y :\in h) \textbf{ where } x :\in g$

   (b) $(g \xrightarrow{a} g',\ a \neq !v)$

   $$\begin{array}{ll} & g \xrightarrow{a} g' \\ \Rightarrow & \{\text{Asym1N}\} \\ & f \textbf{ where } x :\in g \xrightarrow{a} f \textbf{ where } x :\in g' \\ \Rightarrow & \{\text{Asym2}\} \\ & (f \textbf{ where } x :\in g) \textbf{ where } y :\in h \xrightarrow{a} (f \textbf{ where } x :\in g') \textbf{ where } y :\in h \end{array}$$

   And,

   $$\begin{array}{ll} & g \xrightarrow{a} g' \\ \Rightarrow & \{\text{Asym1N}\} \\ & (f \textbf{ where } y :\in h) \textbf{ where } x :\in g \xrightarrow{a} (f \textbf{ where } y :\in h) \textbf{ where } x :\in g' \end{array}$$

   Assumed: $(f \textbf{ where } x :\in g') \textbf{ where } y :\in h \;\sim\; (f \textbf{ where } y :\in h) \textbf{ where } x :\in g'$.

   (c) $(g \xrightarrow{!v} g')$

   $$\begin{array}{ll} & g \xrightarrow{!v} g' \\ \Rightarrow & \{\text{Asym1V}\} \\ & f \textbf{ where } x :\in g \xrightarrow{\tau} [v/x].f \\ \Rightarrow & \{\text{Asym2}\} \\ & (f \textbf{ where } x :\in g) \textbf{ where } y :\in h \xrightarrow{\tau} [v/x].f \textbf{ where } y :\in h \end{array}$$

   And,

$$
\begin{array}{ll}
 & g \stackrel{!v}{\rightarrow} g' \\
\Rightarrow & \{\text{Asym1V}\} \\
 & (f \textbf{ where } y :\in h) \textbf{ where } x :\in g \stackrel{\tau}{\rightarrow} [v/x].(f \textbf{ where } y :\in h)
\end{array}
$$

To see that $[v/x].f \textbf{ where } y :\in h \sim [v/x].(f \textbf{ where } y :\in h)$,

$$
\begin{array}{ll}
 & [v/x].(f \textbf{ where } y :\in h) \\
= & \{\text{substitution distributes}\} \\
 & [v/x].f \textbf{ where } y :\in [v/x].h \\
\Rightarrow & \{h \text{ is } x\text{-free}\} \\
 & [v/x].f \textbf{ where } y :\in h
\end{array}
$$

10. $\mathbf{0} \textbf{ where } x :\in ?k \sim ?k \gg \mathbf{0}$

    The only transition of $?k$ is $?k \stackrel{!v}{\rightarrow} \mathbf{0}$. And, $\mathbf{0}$ has no non-trivial transition.

$$
\begin{array}{ll}
 & ?k \stackrel{!v}{\rightarrow} \mathbf{0} \\
\Rightarrow & \{\text{ASYM1V}\} \\
 & \mathbf{0} \textbf{ where } x :\in ?k \stackrel{\tau}{\rightarrow} [v/x].\mathbf{0}
\end{array}
$$

    And,

$$
\begin{array}{ll}
 & ?k \stackrel{!v}{\rightarrow} \mathbf{0} \\
\Rightarrow & \{\text{SEQ1V}\} \\
 & ?k \gg \mathbf{0} \stackrel{\tau}{\rightarrow} \mathbf{0} \gg \mathbf{0}
\end{array}
$$

    To see $[v/x].\mathbf{0} \sim \mathbf{0} \gg \mathbf{0}$, the lhs is $\mathbf{0}$ and the rhs is bisimilar to $\mathbf{0}$ from $\mathbf{0} \gg f \sim \mathbf{0}$.

11. $\mathbf{0} \textbf{ where } x :\in M \sim M \gg \mathbf{0}$, for any site M.

    The only transition of the constituent expression M is $M \stackrel{M_k()}{\rightarrow} ?k$.

$$
\begin{array}{ll}
 & M \stackrel{M_k()}{\rightarrow} ?k \\
\Rightarrow & \{\text{Asym1N}\} \\
 & \mathbf{0} \textbf{ where } x :\in M \stackrel{M_k()}{\rightarrow} \mathbf{0} \textbf{ where } x :\in ?k
\end{array}
$$

    And,

$$
\begin{array}{ll}
 & M \stackrel{M_k()}{\rightarrow} ?k \\
\Rightarrow & \{\text{Seq1N}\} \\
 & M \gg \mathbf{0} \stackrel{M_k()}{\rightarrow} ?k \gg \mathbf{0}
\end{array}
$$

    Assumed: $\mathbf{0} \textbf{ where } x :\in ?k \sim ?k \gg \mathbf{0}$

Corollaries

(a) f **where** $x :\in M \ \sim \ f \mid M \gg \mathbf{0}$, if f is x-free

$$\begin{array}{ll} & f \textbf{ where } x :\in M \\ \sim & \{\text{proved}\} \\ & (f \mid \mathbf{0}) \textbf{ where } x :\in M \\ \sim & \{f \text{ is } x\text{-free}\} \\ & f \mid (\mathbf{0} \textbf{ where } x :\in M) \\ \sim & \{\text{proved}\} \\ & f \mid M \gg \mathbf{0} \end{array}$$

(b) f **where** $x :\in \mathbf{0} \ \sim \ f$, if f is x-free

$$\begin{array}{ll} & f \textbf{ where } x :\in \mathbf{0} \\ \sim & \{\text{from above}\} \\ & f \mid \mathbf{0} \gg \mathbf{0} \\ \sim & \{\text{proved}\} \\ & f \mid \mathbf{0} \\ \sim & \{\text{proved}\} \\ & f \end{array}$$

We note that the following distributivity law does not hold:

$$\begin{array}{ll} & (f \textbf{ where } x :\in g) \textbf{ where } y :\in h \\ \sim & f \textbf{ where } x :\in (g \textbf{ where } y :\in h), \text{ if } f \text{ is } y\text{-free.} \end{array}$$

B Results about Substitution

Lemma 7 $\langle D.f\rangle = \langle f\rangle \backslash D$, where D is a sequence of substitutions.

Proof: We prove the result for a single substitution c: $\langle c.f\rangle = \langle f\rangle \backslash c$. The lemma follows by repeated application of this result.

$$\begin{array}{ll} & \langle f\rangle \backslash c \\ = & \{\text{definition}\} \\ & \{t \mid ct \in \langle f\rangle\} \\ = & \{f \xrightarrow{c} f' \text{ iff } c.f = f'. \text{ And, } f \xrightarrow{c} f' \stackrel{t}{\Rightarrow} \text{ iff } f' \stackrel{t}{\Rightarrow} \} \\ & \{t \mid t \in \langle c.f\rangle\} \\ = & \{\text{set theory}\} \\ & \langle c.f\rangle \end{array}$$

Lemma 8 Given a sequence of substitutions D and trace sets T_0, T_1, $\cdots$, $(\cup i :: T_i)\backslash D = (\cup i :: T_i \backslash D)$

Proof:

$$\begin{array}{ll} & p \in (\cup i :: T_i)\backslash D \\ \equiv & \{\text{definition of substitution on a trace set}\} \end{array}$$

$$
\begin{array}{ll}
 & Dp \in (\cup i :: T_i) \\
\equiv & \{\text{set theory}\} \\
 & (\exists i :: Dp \in T_i) \\
\equiv & \{\text{definition of substitution on a trace set}\} \\
 & (\exists i :: p \in T_i \backslash D) \\
\equiv & \{\text{set theory}\} \\
 & p \in (\cup i :: T_i \backslash D)
\end{array}
$$

Lemma 9 $S \subseteq T$ implies $S\backslash D \subseteq T\backslash D$, for any sequence of substitutions D and trace sets S and T.

Proof: We prove the result when D has a single substitution c: $S \subseteq T$ implies $S\backslash c \subseteq T\backslash c$. The lemma follows by repeated application of this result.

$$
\begin{array}{ll}
 & p \in S\backslash c \\
\Rightarrow & \{\text{definition of } S\backslash c\} \\
 & cp \in S \\
\Rightarrow & \{S \subseteq T\} \\
 & cp \in T \\
\Rightarrow & \{\text{definition of } T\backslash c\} \\
 & p \in T\backslash c
\end{array}
$$

Lemma 10 $f \sqsubseteq g$ implies $subD.f \sqsubseteq D.g$, for any sequence of substitutions D.

Proof:

$$
\begin{array}{ll}
 & f \sqsubseteq g \\
\Rightarrow & \{\text{definition of } \sqsubseteq \text{ over expressions}\} \\
 & \langle f \rangle \subseteq \langle g \rangle \\
\Rightarrow & \{\text{from Lemma 9}\} \\
 & \langle f \rangle \backslash D \subseteq \langle g \rangle \backslash D \\
\Rightarrow & \{\text{from Lemma 7}\} \\
 & \langle D.f \rangle \subseteq \langle D.g \rangle \\
\Rightarrow & \{\text{definition of } \sqsubseteq \text{ over expressions}\} \\
 & D.f \sqsubseteq D.g
\end{array}
$$

Lemma 11 Suppose $h \xrightarrow{b} h'$. Then for any substitution event c, $c.h \xrightarrow{b} c.h'$, provided b and c are not substitutions to the same variable.

Proof: First, we prove the result if b is a substitution event $[w/y]$.

$h \xrightarrow{[w/y]} [w/y].h$, and
$c.h \xrightarrow{[w/y]} [w/y].c.h = c.[w/y].h$

In the last step, c and $[w/y]$ commute because they are substitutions to different variables.

Henceforth, assume b is not a substitution event. Proof is by structural induction on h. A base expression h which has any transition is of the form

$M(v)$, $?k$ or $let(v)$. Any such expression can make a transition only if it has no variable, i.e., $c.h = h$. Then $h' = c.h'$ and the result holds.

• $h = f \mid g$

Step b is taken by either f or g. Suppose $f \xrightarrow{b} q$. Then,

Proof: $f \mid g \xrightarrow{b} q \mid g$, from $f \xrightarrow{b} q$ and (SYM1)

$c.f \xrightarrow{b} c.q$, induction on $f \xrightarrow{b} q$

$c.f \mid c.g \xrightarrow{b} c.q \mid c.g$, from above and (SYM1)

$c.(f \mid g) \xrightarrow{b} c.(q \mid g)$, substitution

A similar proof applies if $g \xrightarrow{b} q$.

• $h = f >y> g$

We may assume that c is not a substitution to y, because such an event has no effect. Note that only a step of f can result in a step of $f >y> g$. We consider two cases, based on whether b is due to a publication step in f.

1. $f \xrightarrow{!w} q$:

   Proof: $f >y> g \xrightarrow{\tau} q >y> g \mid [w/y].g$
   , from $f \xrightarrow{!w} q$ and (SEQ1V)
   $c.f \xrightarrow{!w} c.q$, induction on $f \xrightarrow{!w} q$
   $c.f >y> c.g \xrightarrow{\tau} c.q >y> c.g \mid [w/y].(c.g)$
   , from above and (SEQ1V)
   $c.f >y> c.g \xrightarrow{\tau} c.q >y> c.g \mid c.[w/y].g$
   , c and $[w/y]$ assign to different variables
   $c.(f >y> g) \xrightarrow{\tau} c.(q >y> g \mid [w/y].g)$
   , substitution rules

2. b is not due to a publication step in f, $f \xrightarrow{b} q$:

   Proof: $f >y> g \xrightarrow{b} q >y> g$, from $f \xrightarrow{b} q$ and (SEQ1N)
   $c.f \xrightarrow{b} c.q$, induction on $f \xrightarrow{b} q$
   $c.f >y> c.g \xrightarrow{b} c.q >y> c.g$, from above and (SEQ1N)
   $c.(f >y> g) \xrightarrow{b} c.(q >y> g)$, substitution rules

• $h = f$ **where** $y :\in g$

We can assume that c is not a substitution to y, because such an event has no effect. Every step of f **where** $y :\in g$ is either because of a step of f or g. We consider three subcases:

1. b is a step of f. That is, $f \xrightarrow{b} q$. **Proof:** f **where** $y :\in g \xrightarrow{b} q$ **where** $y :\in g$
   , from $f \xrightarrow{b} q$ and (ASYM2)
   $c.f \xrightarrow{b} c.q$, induction on $f \xrightarrow{b} q$

$c.f$ **where** $y :\in c.g \xrightarrow{b} c.q$ **where** $y :\in c.g$
, from above and (ASYM2)
$c.(f$ **where** $y :\in g) \xrightarrow{b} c.(q$ **where** $y :\in g)$
, substitution rules

2. b is a step of g not due to a publication. Then $g \xrightarrow{b} q$.

   Proof: f **where** $y :\in g \xrightarrow{b} f$ **where** $y :\in q$
   , from $g \xrightarrow{b} q$ and (ASYM1N)
   $c.g \xrightarrow{b} c.q$, induction on $g \xrightarrow{b} q$
   $c.f$ **where** $y :\in c.g \xrightarrow{b} c.f$ **where** $y :\in c.q$
   , from above and (ASYM1N)
   $c.(f$ **where** $y :\in g) \xrightarrow{b} c.(f$ **where** $y :\in q)$
   , substitution rules

3. b is due to a publication in g (so $b = \tau$). Let $g \xrightarrow{!w} q$.

   Proof: f **where** $y :\in g \xrightarrow{\tau} [w/y].f$, from $g \xrightarrow{b} q$ and (ASYM1V)
   $c.g \xrightarrow{!w} c.q$, induction on $g \xrightarrow{b} q$
   $c.f$ **where** $y :\in c.g \xrightarrow{\tau} [w/y].(c.f)$, from above and (ASYM1V)
   $c.(f$ **where** $y :\in g) \xrightarrow{b} c.([w/y].f)$, substitution rules

Corollary 12 Let t be a trace of f where t has no substitution event for x. Then for any substitution c to x, ct is a trace of f.

Proof: Let s be an execution corresponding to t. We show that cs is an execution of f (and cs corresponds to trace ct). Proof is by induction on the length of s. For $s = \epsilon$, we have $c\epsilon$, i.e., c as an execution of f, by definition of substitution events. For the inductive case, given $s = ar$, **Proof:** $f \xrightarrow{a} f' \stackrel{r}{\Rightarrow}$, for some f'
, ar is an execution of f
a is not a substitution to x , given
$c.f \xrightarrow{a} c.f'$, from Lemma 11
$c.f' \stackrel{r}{\Rightarrow}$, induction
$c.f \stackrel{ar}{\Rightarrow}$, from above two

C Trace Characterizations

C.1 Trace characterization for $(f \mid g)$

Theorem 1: $\langle f \mid g \rangle = \langle f \rangle \mid \langle g \rangle$

Proof:

• $\langle f \mid g \rangle \subseteq \langle f \rangle \mid \langle g \rangle$:
For any trace p of $\langle f \mid g \rangle$, we show a separation (s, t) such that $s \in \langle f \rangle$ and $t \in \langle g \rangle$.

Let q be an execution of $f \mid g$ corresponding to p (i.e., p is obtained by removing all internal events from q). We show that q has a separation (q', q'') where q' is an execution of f and q'' of g. We obtain s and t by removing all internal events from q' and q'', respectively. Then, s and t are traces of f and g, and they form a separation of p. The proof is by induction on the length of q. We use the notation $r \in \langle f \rangle$, for an *execution* r, to denote that the trace corresponding to r is in $\langle f \rangle$.

Case $q = \epsilon$: Let both q' and q'' be ϵ. Now, (ϵ, ϵ) is a separation of ϵ and ϵ is an execution of f (and g) trivially.

Case $q = ar$: We consider two subcases, (1) a is a base event and (2) a is a substitution event.

Subcase, a is a base event: Without loss in generality, assume that $f \stackrel{a}{\rightarrow} f'$, so

$$f \mid g \stackrel{a}{\rightarrow} f' \mid g \stackrel{r}{\Rightarrow} .$$

Applying induction on $f' \mid g \stackrel{r}{\Rightarrow}$, r has a separation (r', r'') where $r' \in \langle f' \rangle$ and $r'' \in \langle g \rangle$. Then, (ar', r'') is a separation of q, from definition. Further, from $f \stackrel{a}{\rightarrow} f' \stackrel{r'}{\Rightarrow}$, $ar' \in \langle f \rangle$, and it is given that $r'' \in \langle g \rangle$.

Subcase, a is a substitution event: Then, $f \mid g \stackrel{a}{\rightarrow} a.f \mid a.g$. Inductively, r has a separation (r', r'') where $r' \in \langle a.f \rangle$ and $r'' \in \langle a.g \rangle$. Define (ar', ar'') as a separation of q; it is easy to verify that (ar', ar'') is indeed a separation. Now, $f \stackrel{a}{\rightarrow} a.f \stackrel{r'}{\Rightarrow}$ and $g \stackrel{a}{\rightarrow} a.g \stackrel{r''}{\Rightarrow}$. Therefore, $ar' \in \langle f \rangle$ and $ar'' \in \langle g \rangle$.

• $\langle f \rangle \mid \langle g \rangle \subseteq \langle f \mid g \rangle$:
Let p be a trace with separation (s, t) where $s \in \langle f \rangle$ and $t \in \langle g \rangle$. We show that $p \in \langle f \mid g \rangle$. Proof is by induction on the length of p.

Case $p = \epsilon$: trivially, $p \in \langle f \mid g \rangle$.

Case $p = ar$: We consider three subcases, (1) a is a base event and (2) a is a substitution event that belongs to both s and t, and (3) a is a substitution event that belongs only to s (the case where the event belongs only to t is symmetric).

Subcase, a is a base event: Since (s, t) is a separation of ar, without loss in generality assume that a is the first event of s, i.e., $s = as'$. Then, from $s \in \langle f \rangle$, $f \stackrel{a}{\Rightarrow} f' \stackrel{s'}{\Rightarrow}$, for some f'. Therefore, r has a separation (s', t) where $s' \in \langle f' \rangle$ and $t \in \langle g \rangle$. Inductively, (since the length of r is smaller than that of p) $r \in \langle f' \mid g \rangle$. Then, $f \mid g \stackrel{a}{\Rightarrow} f' \mid g \stackrel{r}{\Rightarrow}$, i.e., $ar \in \langle f \mid g \rangle$.

Subcase, a is a substitution event that belongs to both s and t:
Then $s = as'$ and $t = at'$. Consider the execution preceding a in s and t, which consist of internal events only. So, for any sequence of internal events τ^*,

Proof: $f \stackrel{\tau^*}{\Rightarrow} f'' \stackrel{a}{\rightarrow} f' \stackrel{s'}{\Rightarrow}$, for some f' and f''
, given $f \stackrel{as'}{\Rightarrow}$
$g \stackrel{\tau^*}{\Rightarrow} g'' \stackrel{a}{\rightarrow} g' \stackrel{t'}{\Rightarrow}$, for some g' and g''
, given $g \stackrel{at'}{\Rightarrow}$
$f \mid g \stackrel{\tau^*}{\Rightarrow} f'' \mid g''$, apply (SYM1, SYM2) on above two
$f'' \mid g'' \stackrel{a}{\rightarrow} f' \mid g'$, apply (SUBST) on $f'' \stackrel{a}{\rightarrow} f'$ and $g'' \stackrel{a}{\rightarrow} g'$
$f \mid g \stackrel{a}{\Rightarrow} f' \mid g'$, from above two (1)
r has a separation (s', t') , (s, t) is a separation of ar, $s = as'$, and $t = at'$
$r \in \langle f' \mid g' \rangle$, induction using above, $s' \in \langle f' \rangle$, and $t' \in \langle g' \rangle$
$f \mid g \stackrel{a}{\Rightarrow} f' \mid g' \stackrel{r}{\Rightarrow}$, from (1), $f \mid g \stackrel{a}{\Rightarrow} f' \mid g'$ and above
That is, $p = ar \in \langle f \mid g \rangle$.

Subcase, a is a substitution event $[v/x]$ that belongs only to s:
Then t has no substitution to x, from the definition of separation. From Corollary 12, $at \in \langle g \rangle$. Hence, p has a separation (s, at), where $s \in \langle f \rangle$, $at \in \langle g \rangle$. The result follows by applying arguments for the previous subcase.

C.2 Trace characterization for $f >x> g$

Lemma 13 Suppose r is a trace with no publications, D is the sequence of substitutions in r, and $f \stackrel{r}{\Rightarrow} f'$. Then $f >x> g \stackrel{r}{\Rightarrow} f' >x> D.g$.

Proof: The subsequence D transforms g to $D.g$ and the remaining events in r have no effect on g.

Theorem 2: $\langle f >x> g \rangle = \langle f \rangle >x> \langle g \rangle$

Proof:

• $\langle f \rangle >x> \langle g \rangle \subseteq \langle f >x> g \rangle$: Suppose $p \in \langle f \rangle >x> \langle g \rangle$, i.e., $p \in (s >x> \langle g \rangle)$, where $s \in \langle f \rangle$. We show that $p \in \langle f >x> g \rangle$. Proof is by induction on the number of publications in s.

Suppose s has no publication: Then $s >x> \langle g \rangle = \{s\}$. Given $p \in \{s\}$, $p = s$. We have to show that $s \in \langle f >x> g \rangle$. This result follows by repeated application of rule (SEQ1N), since s has no publication event.

Suppose $s = r\,!vt$, where r has no publication: Let D be the sequence of substitution events in r. Let $c = [v/x]$. Then,
Proof: $p \in (r\,!vt) >x> \langle g \rangle$, given
$p \in r(t >x> \langle g \rangle \backslash D \mid \langle g \rangle \backslash D \backslash c)$, definition of $(r\,!vt) >x> \langle g \rangle$
$p = rq$ where $q \in (t >x> \langle g \rangle \backslash D \mid \langle g \rangle \backslash D \backslash c)$
, rewriting above
$p = rq$ where $q \in (u \mid \langle g \rangle \backslash D \backslash c)$ and $u \in t >x> \langle g \rangle \backslash D$ (1)
, definition of $\mid$ over sets

$f \stackrel{r}{\Rightarrow} f'' \stackrel{!v}{\rightarrow} f' \stackrel{t}{\Rightarrow}$, from $s \in \langle f \rangle$ and $s = r\,!vt$ (2)
$f >x> g \stackrel{r}{\Rightarrow} f'' >x> D.g$, above, r has no publication and Lemma 13
$f'' >x> D.g \stackrel{\tau}{\rightarrow} f' >x> D.g \mid c.(D.g)$
, from (2) using (SEQ1V)
$f >x> g \stackrel{r}{\Rightarrow} f' >x> D.g \mid c.(D.g)$ (3)
, from above two
$u \in t >x> \langle D.g \rangle$, from (1) and Lemma 7: $\langle g \rangle \backslash D = \langle D.g \rangle$
$t \in \langle f' \rangle$, from (2): $f' \stackrel{t}{\Rightarrow}$
$u \in \langle f' >x> D.g \rangle$, induction on above two; t has fewer publications than s
$q \in \langle f' >x> D.g \rangle \mid \langle c.(D.g) \rangle$, from above, (1) and $\langle g \rangle \backslash D \backslash c = \langle c.(D.g) \rangle$
$q \in \langle f' >x> D.g \mid c.(D.g) \rangle$, Theorem 1 on above
$f >x> g \stackrel{r}{\Rightarrow} f' >x> D.g \mid c.(D.g) \stackrel{q}{\Rightarrow}$
, from (3) and above
$p \in \langle f >x> g \rangle$, from (1): $p = rq$, and above

- Given $p \in \langle f >x> g \rangle$, we show an s such that (1) $s \in \langle f \rangle$, and (2) $p \in s >x> \langle g \rangle$.

Consider the derivation of p in $f >x> g$. At any point during the derivation, we have an expression of the form $f' >x> g' \mid G$. Initially, $f' = f$, $g' = g$ and $G = \mathbf{0}$. The following four rules are applied during the derivation: (SEQ1N), (SEQ1V), (SUBST) and (SYM2), where (SYM2) is applied only for derivations from G. Each of these rules preserves the structure, $f' >x> g' \mid G$.

Given $p \in \langle f >x> g \rangle$, we define the *spine* of a derivation of p as follows. It is a sequence of events, s, created from each application of the first three rules: the non-publication event associated with (SEQ1N) provided it is not τ, the publication event associated with (SEQ1V), and the substitution event associated with (SUBST). That is, we ignore the internal events and all events with (SYM2). Clearly, $s \in \langle f \rangle$. We show $p \in s >x> \langle g \rangle$ by induction on the number of publications in s.

Suppose s has no publication: Then $s >x> g = \{s\}$. We have to show that $p \in \{s\}$, i.e., $p = s$. Since s has no publication, rules (SEQ1V) and, hence, (SYM2) are never applied during the derivation of p. Therefore, $p = s$.

Suppose $s = r\,!vt$: Let D be the sequence of substitutions in r.

Proof: $f \stackrel{r}{\Rightarrow} f'' \stackrel{!v}{\rightarrow} f' \stackrel{t}{\Rightarrow}$, for some f' and f''
, from $r\,!vt \in \langle f \rangle$
$f >x> g \stackrel{r}{\Rightarrow} f'' >x> D.g$, Observation 1
$f'' >x> D.g \stackrel{\tau}{\rightarrow} f' >x> D.g \mid c.(D.g)$
, from $f'' \stackrel{!v}{\rightarrow} f'$
$f >x> g \stackrel{r}{\Rightarrow} f' >x> D.g \mid c.(D.g)$
, from above two
$p \in r(q \mid \langle c.(D.g) \rangle)$, where $q \in \langle f' >x> D.g \rangle$ (1)
, from above and the spine construction

t is a spine of q, where $q \in f' >x> D.g$
, from the construction of s and $s = r\,!vt$
, SYM2 is applied only for expansion of $c.(D.g)$
$q \in t >x> \langle D.g\rangle$, induction on above; t has fewer publications than s
$p \in r(t >x> \langle D.g\rangle \mid \langle c.(D.g)\rangle)$
, from (1) and above using definition of $\mid$ over sets
$p \in r(t >x> \langle g\rangle\backslash D \mid \langle g\rangle\backslash D\backslash c)$, Lemma 7: $\langle D.g\rangle = \langle g\rangle\backslash D$ and $\langle c.(D.g)\rangle = \langle g\rangle\backslash D\backslash c$
$p \in (r\,!vt) >x> \langle g\rangle$, definition of $(r\,!vt) >x> \langle g\rangle$
$p \in s >x> \langle g\rangle$, $s = r\,!vt$

C.3 Trace characterization for $(f$ **where** $x :\in g)$

Theorem 3: $\langle f$ **where** $x :\in g\rangle = \langle f\rangle$ **where** $x :\in \langle g\rangle$

Proof:
The theorem is equivalent to $\langle f$ **where** $x :\in g\rangle = (A \cup B)$, where

$$A = \{p \mid p \in (s \mid r),\ s \in \langle f\rangle,\ r \in \langle g\rangle,\ r \text{ has no publication}\}$$

$$B = \{p \mid p \in (s \mid r)t,\ r \text{ has no publication},\ (\exists v :: r\,!v \in \langle g\rangle, s[v/x]t \in \langle f\rangle)\}$$

• $\langle f$ **where** $x :\in g\rangle \subseteq (A \cup B)$:
Let $p \in \langle f$ **where** $x :\in g\rangle$. We consider two cases based on whether rule (ASYM1V), which handles publications by g, is ever used in the derivation of p.

Case 1) Rule (ASYM1V) is never used in the derivation of p: We show that $p \in A$. Since only ASYM1N and ASYM2 are used, the case is analogous to that for $\mid$; therefore, $p \in (s \mid r)$ where $s \in \langle f\rangle$ and $r \in \langle g\rangle$. And, r has no publication because rule (ASYM1V) is never used.

Case 2) Rule (ASYM1V) is used in the derivation of p: We show that $p \in B$. A derivation in which rule (ASYM1V) is used is of the following form.

$$f \textbf{ where } x :\in g \overset{q}{\Rightarrow} f' \textbf{ where } x :\in g' \overset{\tau}{\rightarrow} [v/x].f' \overset{t}{\Rightarrow} \text{ where } g' \overset{!v}{\rightarrow}$$

Thus, $p = qt$. Since (ASYM1V) is never used in the derivation of q, from Case 1, $q \in (s \mid r)$, and r has no publication. Further, $f \overset{s}{\Rightarrow} f'$ and $g \overset{r}{\Rightarrow} g'$. Therefore,

Proof: $p = qt \in (s \mid r)t$, from $q \in (s \mid r)$ (1)
r has no publication event , from above discussion (2)
$r\,!v \in \langle g\rangle$, from $g \overset{r}{\Rightarrow} g' \overset{!v}{\rightarrow}$ (3)

$f' \stackrel{[v/x]}{\rightarrow} [v/x].f' \stackrel{t}{\Rightarrow}$, from $[v/x].f' \stackrel{t}{\Rightarrow}$
$s[v/x]t \in \langle f \rangle$, from $f \stackrel{s}{\Rightarrow} f'$ and above (4)
We conclude $p \in B$ from (1, 2, 3, 4).

• $(A \cup B) \subseteq \langle f \textbf{ where } x :\in g \rangle$:

Case 1) $p \in A$: We show that $p \in \langle f \textbf{ where } x :\in g \rangle$, given $p \in (s \mid r)$, where $s \in \langle f \rangle$, $r \in \langle g \rangle$, and r has no publication event. Then, rule (ASYM1V) is never used in this derivation. We use arguments similar to those for symmetric composition, $\mid$, for this proof.

Case 2) $p \in B$: We show that $p \in \langle f \textbf{ where } x :\in g \rangle$.

Proof: $f \stackrel{s}{\Rightarrow} f'' \stackrel{[v/x]}{\rightarrow} f' \stackrel{t}{\Rightarrow}$, for some f'' and f' (1)
, from $s[v/x]t \in \langle f \rangle$
$g \stackrel{r}{\Rightarrow} g'' \stackrel{!v}{\rightarrow}$, from $r\,!v \in \langle g \rangle$ (2)
$f \textbf{ where } x :\in g \stackrel{q}{\Rightarrow} f'' \textbf{ where } x :\in g''$, and $q \in (s \mid r)$
, from (1, 2) and Case 1: r has no publication
$f'' \textbf{ where } x :\in g'' \stackrel{\tau}{\rightarrow} f'$, apply (ASYM1V) on (1) and (2)
$f' \stackrel{t}{\Rightarrow}$, from (1)
$f \textbf{ where } x :\in g \stackrel{q}{\Rightarrow} f'' \textbf{ where } x :\in g'' \stackrel{\tau}{\rightarrow} f' \stackrel{t}{\Rightarrow}$
, combining above three
$f \textbf{ where } x :\in g \stackrel{p}{\Rightarrow}$, from above and $p = qt$

D Monotonicity and Continuity

D.1 Monotonicity of Orc combinators

Lemma 14 $T \subseteq R$ implies $s >x> T \subseteq s >x> R$

Proof: The proof is by induction on the number of publications in s.

• s has no publication: Then both $s >x> T$ and $s >x> R$ are $\{s\}$.

• $s = r\,!vt$, where r has no publication: Let D be the sequence of substitution events in r.

$$
\begin{array}{ll}
 & s >x> T \\
= & \{s = r\,!vt\} \\
 & (r\,!vt) >x> T \\
= & \{\text{definition of } (r\,!vt) >x> T\} \\
 & r(t >x> T\backslash D \mid T\backslash D\backslash[v/x]) \\
\subseteq & \{\text{from Lemma 9, } T\backslash D \subseteq R\backslash D\text{; induction and monotonicity of } \mid\} \\
 & r(t >x> R\backslash subD \mid T\backslash D\backslash[v/x]) \\
\subseteq & \{\text{from Lemma 9, } T\backslash D\backslash[v/x] \subseteq R\backslash D\backslash[v/x]\text{; monotonicity of } \mid\}
\end{array}
$$

$$
\begin{array}{ll}
 & r(t >x> R\backslash D \mid R\backslash D\backslash[v/x]) \\
= & \{\text{definition of } (r\,!vt) >x> R\} \\
 & (r\,!vt) >x> R \\
= & \{s = r\,!vt\} \\
 & s >x> R
\end{array}
$$

Monotonicity in the Left Argument Each Orc combinator is monotonic in its left argument, which follows from Lemma 4, part(1). Specifically, given $f \sqsubseteq g$, we show $f \mid h \ \sqsubseteq\ g \mid h$. Proofs for the other combinators follow the same pattern. From Lemma 4,

$$
\begin{array}{ll}
 & (\langle f\rangle \mid \langle h\rangle) \cup (\langle g\rangle \mid \langle h\rangle) = (\langle f\rangle \cup \langle g\rangle) \mid \langle h\rangle \\
\Rightarrow & \{f \sqsubseteq g \text{ implies } \langle f\rangle \subseteq \langle g\rangle\} \\
 & (\langle f\rangle \mid \langle h\rangle) \cup (\langle g\rangle \mid \langle h\rangle) = \langle g\rangle \mid \langle h\rangle \\
\Rightarrow & \{\text{set theory}\} \\
 & (\langle f\rangle \mid \langle h\rangle) \subseteq (\langle g\rangle \mid \langle h\rangle) \\
\Rightarrow & \{\text{Theorem 1}\} \\
 & \langle f \mid h\rangle \subseteq \langle g \mid h\rangle \\
\equiv & \{\text{definition of } \sqsubseteq \text{ over expressions}\} \\
 & f \mid h \ \sqsubseteq\ g \mid h
\end{array}
$$

Monotonicity in the Right Argument Monotonicity in the right argument for $(f \mid g)$ and $(f \textbf{ where } x :\in g)$ follow from Lemma 4, part(2). We need a special proof for $f >x> g$, which is similar to the other proofs; it employs Lemma 14.

$$
\begin{array}{ll}
 & \langle h >x> f\rangle \\
= & \{\text{Theorem 2}\} \\
 & \langle h\rangle >x> \langle f\rangle \\
\subseteq & \{\text{given } \langle f\rangle \subseteq \langle g\rangle, \text{ from Lemma 14}\} \\
 & \langle h\rangle >x> \langle g\rangle \\
= & \{\text{Theorem 2}\} \\
 & \langle h >x> g\rangle
\end{array}
$$

D.2 Continuity of Orc combinators

Characterization of the Least Upper Bound

Theorem 15 $\langle \sqcup f\rangle = (\cup i :: \langle f_i\rangle)$.

Proof: Let F be such that $\langle F\rangle = (\cup i :: \langle f_i\rangle)$. We show $F = (\sqcup f)$.
F is an upper bound: **Proof:** $\langle F\rangle = (\cup i :: \langle f_i\rangle)$, given
$\langle f_i\rangle \subseteq \langle F\rangle$, for any i , from above
$f_i \sqsubseteq F$, for any i , definition of $\sqsubseteq$ over expressions
F is an upper bound , definition of upper bound
F is the least upper bound, i.e., for any upper bound G, $F \sqsubseteq G$: **Proof:**
$f_i \sqsubseteq G$, for any i , G is an upper bound of f

$\langle f_i \rangle \subseteq \langle G \rangle$, for any i , from above
$(\cup i :: \langle f_i \rangle) \subseteq \langle G \rangle$, set theory
$\langle F \rangle \subseteq \langle G \rangle$, given $\langle F \rangle \;=\; (\cup i :: \langle f_i \rangle)$
$F \sqsubseteq G$, definition of $\sqsubseteq$ over expressions $\Box$

Lemma 16 $(S >x> (\cup i :: T_i)) \subseteq (\cup i :: S >x> T_i)$, for any trace set S, and a chain of trace sets $T_0 \subseteq T_1 \cdots$.

Proof: We prove the result for a single trace s instead of a set S. Using the definition of $>x>$ over trace sets, the result follows.

We use R as an abbreviation for $(\cup i :: T_i)$. Proof is by induction on the number of publications in s. If s has no publication, $s >x> R = \{s\}$ and $(\cup i :: s >x> T_i) = (\cup i :: \{s\}) = \{s\}$.

Assume that $s = r\,!vt$ where r has no publication. Let D be the sequence of substitution events in r.

$$
\begin{array}{ll}
 & (r\,!vt) >x> (\cup i :: T_i) \\
= & \{R = (\cup i :: T_i)\} \\
 & (r\,!vt) >x> R \\
= & \{\text{definition, let } c = [v/x]\} \\
 & r(t >x> R\backslash D \mid R\backslash D\backslash c) \\
= & \{\text{rewrite}\} \\
 & r(u \mid u'),\ \text{where } u \in t >x> R\backslash D,\ u' \in R\backslash D\backslash c \\
= & \{\text{from Lemma 8, } R\backslash D = (\cup i :: T_i\backslash D),\ R\backslash D\backslash c = (\cup i :: T_i\backslash D\backslash c)\} \\
 & r(u \mid u'),\ \text{where } u \in t >x> (\cup i :: T_i\backslash D),\ u' \in (\cup i :: T_i\backslash D\backslash c) \\
\subseteq & \{\text{induction on } u \in t >x> (\cup i :: T_i\backslash D);\ t \text{ has fewer publications than } s\} \\
 & r(u \mid u'),\ \text{where } u \in (\cup i :: t >x> T_i\backslash D),\ u' \in (\cup i :: T_i\backslash D\backslash c) \\
= & \{u \in (\cup i :: t >x> T_i\backslash D) \text{ means } u \in t >x> T_m\backslash D,\ \text{for some } m\} \\
 & r(u \mid u'),\ \text{where } u \in t >x> T_m\backslash D,\ u' \in (\cup i :: T_i\backslash D\backslash c) \\
= & \{u' \in (\cup i :: T_i\backslash D\backslash c) \text{ implies } u' \in T_n\backslash D\backslash c, \text{ for some } n\} \\
 & r(u \mid u'),\ \text{where } u \in t >x> T_m\backslash D,\ u' \in T_n\backslash D\backslash c \\
\subseteq & \{\text{let } i = \max(m,n), \text{ i.e., } T_m \subseteq T_i \text{ and } T_n \subseteq T_i.\ \text{Use monotonicity}\} \\
 & r(u \mid u'),\ \text{where } u \in t >x> T_i\backslash D,\ u' \in T_i\backslash D\backslash c \\
= & \{\text{definition of } >x> \text{ over sets}\} \\
 & r(t >x> T_i\backslash D \mid T_i\backslash D\backslash c) \\
= & \{\text{definition}\} \\
 & (r\,!vt) >x> T_i \\
\subseteq & \{\text{set theory}\} \\
 & (\cup i :: (r\,!vt) >x> T_i) \qquad \Box
\end{array}
$$

$>x>$ is continuous in its right argument.

Lemma 17 Given a sequence g, and $h_i = f >x> g_i$, for all i,
$(\sqcup h) \;\cong\; f >x> (\sqcup g)$, i.e., $\langle (\sqcup h) \rangle = \langle f >x> (\sqcup g) \rangle$.

• $(\sqcup h) \;\sqsubseteq\; f >x> (\sqcup g)$: **Proof:** $f >x> g_i \sqsubseteq f >x> (\sqcup g)$, $g_i \sqsubseteq (\sqcup g)$ and the monotonicity of $>x>$
$\sqcup(f >x> g_i) \sqsubseteq f >x> (\sqcup g)$, definition of least upper bound

• $f >x> (\sqcup g) \sqsubseteq (\sqcup h)$:

$$
\begin{array}{ll}
 & \langle f >x> (\sqcup g)\rangle \\
= & \{\text{Theorem 2}\} \\
 & \langle f\rangle >x> \langle \sqcup g\rangle \\
\subseteq & \{\text{Lemma 16 from the appendix}\} \\
 & (\sqcup i :: \langle f\rangle >x> \langle g_i\rangle) \\
= & \{\text{Theorem 2}\} \\
 & (\sqcup i :: \langle f >x> g_i\rangle) \\
= & \{\text{definition of } h_i\} \\
 & (\sqcup i :: \langle h_i\rangle) \\
= & \{\text{definition of lub}\} \\
 & \langle \sqcup h\rangle
\end{array}
$$

E Proof of $f >x> let(x) \simeq f$

We show that $f >x> let(x) \cong f$. Clearly this result can not be proved by strong bisimulation since the two expressions have different executions. But their traces are identical. We use induction on the structure of f.

In proving $f \cong g$ where both f and g are closed, we ignore substitution events, because no substitution event has any effect on a closed expression.

• $let(v) >x> let(x) \cong let(v)$: **Proof:** $let(v) \stackrel{!v}{\rightarrow} \mathbf{0}$, apply rule (LET)
$let(v) >x> let(x) \stackrel{\tau}{\rightarrow} \mathbf{0} >x> let(x) \mid let(v)$
, apply rule (SEQ1V)
$let(v) >x> let(x) \stackrel{\tau}{\rightarrow} let(v)$, $\mathbf{0} >x> let(x) \cong \mathbf{0}$ and $\mathbf{0} \mid let(v) \cong let(v)$ (Section 4.1)

Therefore, the set of executions of $let(v) >x> let(x)$ is $\tau\langle let(v)\rangle$ (ignoring the substitution events). So, its set of traces is $\langle let(v)\rangle$.

• $?k >x> let(x) \cong ?k$: **Proof:** $?k \stackrel{k?v}{\rightarrow} let(v)$, for all values v
, apply rule (SITERET)
$?k >x> let(x) \stackrel{k?v}{\rightarrow} let(v) >x> let(x)$
, apply rule (SEQ1N)
$?k >x> let(x) \stackrel{k?v}{\rightarrow} let(v)$, $let(v) >x> let(x) \cong let(v)$, from the previous proof

Therefore, the traces of $?k$ and $?k >x> let(x)$ are both $k?v\langle let(v)\rangle$, for all values v.

• $M(v) >x> let(x) \cong M(v)$: **Proof:** $M(v) \stackrel{M_k(v)}{\rightarrow} ?k$, for some u , apply rule (SITECALL)
$M(v) >x> let(x) \stackrel{M_k(v)}{\rightarrow} ?k >x> let(x)$
, apply rule (SEQ1N)
$M(v) >x> let(x) \stackrel{M_k(v)}{\rightarrow} ?k$, $?k >x> let(x) \cong ?k$, from the previous proof

Therefore, the traces of $M(v)$ and $M(v) >x> let(x)$ are both $M_k(v)\langle ?k\rangle$, for any k.

• $M(y) >x> let(x) \cong M(y)$: **Proof:** $M(y) \stackrel{[v/y]}{\rightarrow} M(v)$, for all v , apply rule (SUBST)
$M(y) >x> let(x) \stackrel{[v/y]}{\rightarrow} M(v) >x> let(x)$
, apply rule (SEQ1N)
$M(v) >x> let(x) \stackrel{[v/y]}{\rightarrow} M(v)$, $M(v) >x> let(x) \cong M(v)$, from the previous proof

Therefore, the traces of $M(y)$ and $M(y) >x> let(x)$ are both $[v/y]\langle M(v)\rangle$, for all v.

• $(f \mid g) >x> let(x) \cong (f \mid g)$:

$$
\begin{array}{ll}
 & (f \mid g) >x> let(x) \\
\cong & \{\text{Distributivity law from Section 4.1}\} \\
 & f >x> let(x) \mid g >x> let(x) \\
\cong & \{\text{induction on } f >x> let(x) \text{ and } g >x> let(x)\} \\
 & f \mid g
\end{array}
$$

• $(f >y> g) >x> let(x) \cong (f >y> g)$:

$$
\begin{array}{ll}
 & (f >y> g) >x> let(x) \\
\cong & \{\text{Associativity law from Section 4.1; } let(x) \text{ is } y\text{-free}\} \\
 & f >y> (g >x> let(x)) \\
\cong & \{\text{induction on } g >x> let(x)\} \\
 & f >y> g
\end{array}
$$

• $(f \textbf{ where } y :\in g) >x> let(x) \cong (f \textbf{ where } y :\in g)$:

$$
\begin{array}{ll}
 & (f \textbf{ where } y :\in g) >x> let(x) \\
\cong & \{\text{Distributivity law from Section 4.1; } let(x) \text{ is } y\text{-free}\} \\
 & (f >x> let(x)) \textbf{ where } y :\in g \\
\cong & \{\text{induction on } f >x> let(x)\} \\
 & f \textbf{ where } y :\in g
\end{array}
$$

E.1 Simpler proof of base cases

Proof summary: Observe that all traces of base expressions are of the form $r\,!v$, where r has no publications. By trace characterization we can show that $r\,!v >x> \langle let(x)\rangle = \{r\,!v\}$ for all traces of this form. This establishes the base case.